ADVANCED INDUSTRIAL CONTROL TECHNOLOGY

PENG ZHANG

Advanced Industrial Control Technology

Peng Zhang

AMSTERDAM • BOSTON • HEIDELBERG • LONDON • NEW YORK • OXFORD
PARIS • SAN DIEGOSAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

William Andrew is an imprint of Elsevier

William Andrew is an imprint of Elsevier
The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, UK
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA

First edition 2010

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is availabe from the Library of Congress

ISBN 13: 978 1 4377 7807 6

For information on all Elsevier publications visit our web site at books.elsevier.com

Printed and bound in the United Kingdom

10 11 12 13 14 15 10 9 8 7 6 5 4 3 2 1

Preface

The first book of mine, entitled Industrial Control Technology: A Handbook for Engineers and Researchers, was published in June of 2008. A large number of engineers and researchers working in the processing, chemical and manufacturing industries think that the book is an applicable, implemental reference for modern real-time and distributed industrial control technologies - due to its comprehensive coverage, sophisticated explanation, and detailed content. That book also attracts great interest in universities and colleges around the world, which encouraged me to develop a book to be used for training professional engineers and teaching diploma courses in the field of industrial process control and production automation.

Nowadays, global industries find it more and more necessary to recruit engineers and graduates who have obtained solid training in technical skills and engineering expertise, rather than just the theory and strategy of industrial controls from universities or colleges. To offer this professional training and diploma courses, worldwide universities and colleges seem to urgently need an excellent book which provides full coverage of industrial control technologies suitable for the implementation of modern industrial control engineering. This book, Advanced Industrial Control Technology, aims at meeting this need.

This book fully covers industrial control technical topics with the intention of documenting all the key technologies applicable for various industrial control systems and engineering; after completing the courses within this book, engineers and students should be capable of implementing most control systems and engineering in different industrial applications.

This book provides a full series of the industrial control technologies traced through industrial control developments, including the traditional control techniques; and modern control techniques using real-time, distributed, robotic, embedded, computer and wireless control technologies.

This book gives a complete profile of these technologies from the field layer and the control layer to the operator layer for industrial control systems. This book includes all the interfaces in industrial control systems: the interfaces between controllers and systems; the interfaces between different layers; the interfaces between operators and systems. It is very noteworthy that this book not only describes the details of both real-time operating systems and distributed operating systems, but also of the microprocessor boot code, which seems beyond the scope of most industrial control books.

For the components, devices and hardware circuits, this book emphasizes the technical issues together with their working principles and physical mechanisms. The technical issues include the specification parameters, installation procedures, and calibration and configuration methodologies, etc. For the software packages, this book gives the programing methods in additional to their semantics and rationales.

Therefore for all industrial control and automation technologies, this book can serve equally as a textbook for postgraduates and undergraduates; as a guidebook for engineering and technical training; and as a handbook for practical engineers and researchers.

BACKGROUND

In the nineteenth century during the Industrial Revolution in Great Britain, machines were invented to replace human hands in the production of things such as textiles and tools. Industry emerged in the

world as an important sector within human society. Accordingly, production and motion processes gradually began to be carried out by means of machines rather than human hands. Thereafter, to free human eyes and brains as much as possible from operating and monitoring various machine processes, Industrial Control and Automation was generated as a special technology used in industry. In 1910, Henry Ford achieved automation in his automobile assembly plant. Then, Allen Rogers and Arthur D. Little in America put forward the "Unit Operation" concept and the technique led to continuous chemical processing and extensive automation during the 1920s. With these pioneering works, "Industrial Control and Automation" came to be more and more used in various industries, and thus underwent significant development from the early part of the twentieth century.

In the 1930s, Harry Nyquist put forward a stability criterion by expanding the mathematical dynamic system and stability theory of Alexander Lyapunov, which initiated control theory. Thereafter in World War II, as an approach to fire control, airplane guidance, and large-scale chemical production control, the "system" concept was introduced to industrial control and automation, which led control theory to a higher level.

In 1947, the first generation of modern programed electronic computers was built in America. In 1968, a digital computer was produced to compete with electromechanical systems then under development for the main flight control computer in the US Navy. Based on computer technology, control engineers developed programable controllers and embedded systems. The first recognizably modern embedded system was the Apollo Guidance Computer, developed by Charles Stark Draper at the MIT Instrumental Laboratory.

In the 1970s, a great technical revolution occurred in the electronic and semiconductor industries, in which the integrated circuits were generated. By means of large-scale integrated circuit technologies, all the central-processing functions of a computer could be built on a single semiconductor chipset called a microprocessor or microcontroller.

Since the 1970s, Intel, IBM and other super-corporations have kept advancing the techniques of microprocessors to make their bit capacity increase from 4-bit, 8-bit, 16-bit to 32-bit in 1979. The highest capacities of microprocessor at present are 64-bit and 128-bit, developed by the AMD and Cyrix corporations, respectively, in the early 1990s. Nowadays, the single microprocessor has been extended into the multi-core microprocessor using parallelism and shared mechanisms.

- With the enhancement of the capacities and speeds of microprocessors, a widespread use of embedded control became feasible in the world from the mid-1990s. Accordingly, an important standard of industrial control was created, which is real-time control.
- Another high-technology achievement of the twentieth century is computer communication networks, which were developed in the 1980s. Inspired by the computer communication networks, distributed control was developed as another powerful control technique.

After World War II, high technology advanced throughout the world, and created many technical miracles such as programable integrated circuits, supercomputers and computer networks, etc. As a result, automatic controls in many vehicles, such as in aircraft, ships and cars, have greatly increased their scope; production automation in many industries, for example, in coal mines, steel plants and chemical factories, reached a higher level. Furthermore, more and more countries in the world are capable of making advanced space vehicles such as satellites and spacecraft that have very high requirements of the automatic controls inside them.

After decades of development, industrial control has formed a complete category with two aspects: industrial control theory and industrial control technology.

- Industrial control theory is the design guide for industrial control engineering. Industrial control theory includes the basic mathematical theory, control system theory, control models, control algorithms, and control strategies.
- Industrial control technology is the implemental guide for industrial control engineering. Industrial control technology includes field-level devices, embedded controllers and computer hardware, the components and protocols of industrial networks, and the software, mainly composed of real-time operating systems, distributed operating systems, embedded application software and control system routines.

Overall, with more than two centuries' development and evolution, industrial control and automation technologies have so advanced that they benefit us in all aspects of our life and in all kinds of production systems; they are closely integrated with computer hardware and software, network devices and communication technologies; they are faithfully based on modern results in the mathematical and physical sciences.

ORGANIZATION

This book has been structured into parts, chapters, sections, subsections and titled paragraphs, etc. There are eight parts in this book. Each part comprises several chapters. In total, this book consists of nineteen chapters within these eight parts. Each chapter in this book is dedicated to one special topic, and contains a number of sections explaining an aspect of this topic. Every chapter has two appended lists: one is a list of Problems and the other is Further Reading. The problem lists provide readers with questions for reviewing the knowledge included in the chapter. The Further Reading points readers to further sources related to the chapter.

Part 1 is titled "Industrial control fundamentals". This part contains two chapters: Chapter 1 is "Industrial control systems", and Chapter 2 is "Industrial control engineering". Modern control systems fall into three types described in Chapter 1: embedded control systems refer to a special relationship of control system to controlled system; real-time control systems are those control systems which satisfy the temporal standard for control operations; distributed control systems describe control systems that have special architectures and rationales in delivering control functions. In industry, the subjects of control applications can be processes such as a chemical reaction or an electrical transmission, a motion such as a car's motion or an aircraft's motion, a production such as making a machine or producing clothes, etc. Chapter 2 discusses the three types of modern industrial control engineering: process control, motion control and production automation.

Part 2, entitled "Field elements of industrial control systems", includes Chapters 3 and 4. In industrial control systems, the field level means the lowest level where control agents directly detect, measure, drive, and apply forces to the controlled objects. Chapter 3, entitled "Sensors and actuators", describes the sensors and actuators of the field-level elements, including optical sensors, temperature and distance sensors, and measurement sensors; and electric, magnetic, pneumatic, hydraulic and piezoelectric actuators. Chapter 4, entitled "Transducers and valves", describes transducers and valves

including limited switches, photoelectric switches, proximity switches, ultrasonic transducers, linear and rotary motors, control valves, solenoid valves, and float and flow valves.

Part 3 of this book is entitled "Embedded hardware in industrial control systems". This part provides a detailed list of the types of microelectronic components used in control systems. These are the "Microprocessors" in Chapter 5, and the "Programmable-logic and application-specific integrated circuits (PLASIC)" in Chapter 6. Chapter 5 describes both single-core microprocessor and multi-core microprocessor units, emphasizing the multi-core microprocessors because this type of microelectronic architecture is much more powerful than single-core for performing real-time and distributed controls due to their parallelism and the shared mechanism. Chapter 6 introduces the three main types of ASICs: FPGA, MPGA and PLD. In this chapter, several typical ASIC devices are described, including programmable peripheral I/O ports, programmable interrupt controllers, programmable timers, and CMOS and DMA controllers.

Part 4 is entitled "Controllers and computers for industrial controls". Each of the controllers discussed in this part, similar to computers, is a system with its own hardware and software so that it is able to independently perform control functions. In Chapter 7, it explains the industrial intelligent controllers that are necessary for both industrial production control and industrial process control; these are PLC controllers, CNC controllers, and fuzzy-logic controllers. In Chapter 8, it explains some industrial process controllers, including PID controllers, batch process controllers and servo motion controllers. Industrial computers are specially architectured computers used in industrial control systems playing similar functions to the controllers. Chapter 9 of this book is on "Industrial computers" and explains industrial motherboards, industrial personal computers (PC), and some computer peripherals and accessories.

The device networks in industrial control systems are basically fieldbus-based multiple-node networks that intelligently connect field elements, controllers, computers and control subsystems for applying process, motion or production controls to industrial and enterprise systems. These networks demand interoperability of control functions at different hierarchical levels with digital data communications. Part 5 of this book is dedicated to "Embedded networks in industrial control systems". In this part, Chapter 10 introduces the layer model, architectures, components, functions and applications of several primary industrial control networks: CAN, SCADA, Ethernet, DeviceNet, LAN, and other enterprise networks. Chapter 11 of this part deals with networking devices, including networking hubs, switches, routers, bridges, gateways and repeaters. This chapter also provides some key techniques used in these networking devices.

Part 6 is on "Interfaces in industrial control systems". This part describes three types of interfaces; field interfaces in Chapter 12, human machine interfaces in Chapter 13, and data transmission interfaces in Chapter 14. These three types of interfaces basically cover all the interface devices and technologies existing in the various industrial control systems. The actuator-sensor interface locates at the front or rear of the actuator-sensor level to bridge the gap between this level and the controllers. The HART was a traditional field subsystem but is still popular at present because it integrates digital signals with analogue communication channels. Fieldbuses, such as Foundation and Profibus, are actually dedicated embedded networks at the field level. The data transmission interfaces include the transmission control and I/O devices that are used for connecting and communicating with controllers. The human machine interfaces contain both the tools and technologies to provide humans with easy and comfortable use of the technology.

The title of Part 7 is "Embedded industrial control software". Modern control technology features computer hardware, communication networks and embedded software. Embedded software for control purpose includes three key components: the "Microprocessor boot code" in Chapter 15, the "Real-time operating systems" in Chapter 16, and the "Distributed operating systems" in Chapter 17. These chapters provide the basic rationale, semantics, principles, work sequence and program structures for each of these three components. Chapter 15 explains the firmware of a microprocessor chipset. Chapter 16 gives all the details of real-time operating systems, which are the platforms needed for a control system to satisfy real-time criteria. Chapter 17 explains the necessary platform for distributed control systems, which is the distributed operating system.

Part 8 is for the profound topic of "Industrial control system routines". This part is dedicated to system routines that make the control systems more efficient, more user-friendly and safe to operate. Chapter 18 explains industrial system operation routines, including the self-test routines at power-on and power-down, installation and configuration routines, diagnostic routines, and calibration routines. Chapter 19, the last chapter of this book, is on industrial control system simulation routines, which are process modeling and simulation routines, manufacture modeling and simulation routines, and the simulator, toolkits and toolboxes for implementing simulation. In Chapter 19, the identification principles and techniques for model-based control are also discussed.

SUGGESTIONS

This book is an engineering- and implement-oriented technical textbook, guidebook and handbook, which hence requires not only classrooms but also laboratories. Field practices, including field experiments and implementations, are necessary for learning the material in this book. In places where conditions are ready, the training in each part of this book should be based on one or several prototype industrial systems. For example, the following industrial systems and networks can provide the background for studying this book: manufacturing industries, including the makers of machines, motors, vehicles, cars, aircrafts, ships, and so on; chemistry production industries, including the producers of medicine, plastics, fibers, and so on; energy industries, including electric power production and transmission, crude oil fields, coal mines, and so on; transport industries, including railways, underground transport, city roads, inner-river shipping, and so on; food production industries, including the makers of breads, sugar, beers, wines, and so on; device makers for industrial control networks such as industrial Ethernet, Fieldbuses, industrial computers; industrial control system interfaces such as actuator-sensor interfaces, human machine interfaces, and data transmission interfaces, etc.

This book contains so much that it could take one and a half academic years to complete the study. Therefore, depending on your requirements, selection of which contents and topics in this book to read is very feasible. Because this book consists of eight parts, each part can be used for a specific area of study, such as: industrial control systems and engineering fundamentals; sensors, actuators, transducers and valves in industrial control systems and engineering; embedded hardware in industrial control engineering: single-core and multi-core microprocessors; Application-specific integrated circuits (ASIC) and programmable intelligent devices; digital controllers in industrial control systems; industrial computers; industrial control networks: architectures and components and interfaces;

industrial control networks: communication protocols and software; the buses and routers in industrial control networks; real-time operating systems; distributed operating systems; microprocessor boot code; industrial control system routines; industrial processes and systems modeling; simulating the industrial control system; etc.

SOURCES

Writing this book has involved reference to a myriad of sources including academic and technical books, journal articles, lecture notes, and in particular industry technical manuals, device specifications and company introductory or demonstration documents etc. displayed on websites of various dates and locations. The number and the scale of the sources are so huge that it would be practically impossible to acknowledge each source individually in the body of the book. The sources for each chapter of this book are therefore placed at the end of each chapter. This method has two benefits. It enables the author to acknowledge the contribution of other individuals, institutions and companies whose scholarship or products have been referred to in this book; and it provides help for the reader who wishes to read further on the subject.

Acknowledgments

My most sincere thanks go to my family: to my wife Minghua Zhao, and my son Huzhi Zhang, for their unwavering understanding and support.

I would also like to thank the anonymous reviewers for their valuable comments and suggestions on the book manuscript. Without their contribution and support, there would not have been this book.

CHAPTER

Industrial control systems 1

In the nineteenth century, during the Industrial Revolution in Great Britain, machines were invented to mechanize the production of textiles and tools, and other such items. Industry emerged world as an important sector of human society. Production and transport processes gradually began to be carried out by machines rather than by human hands. In order to free human eyes and brains as much as possible from operating and monitoring machine processes, Industrial Control and Automation developed as industrial technology.

In the 1980s, industrial control systems only encompassed supervisory control and data acquisition (SCADA) systems, and programmable logic controllers (PLC). As microprocessors and programmable integrated circuits developed in the 1990s, industrial control systems began to incorporate computers. Computerized control systems are powerful and efficient, and thus have found more and more applications across many industries, such as electricity, water, oil and gas, chemical, transportation, and manufacturing. Computerized control systems are different from computer control systems. In a computer control system, the computer takes the role of a supervisor separate from the controlled objects. In contrast, a computerized control system incorporates hardware and software into the system to be controlled, thus creating a single unified system. In order to differentiate, the term the two "embedded control" is used for computerized control systems.

There are two important types of embedded controls; real-time control and distributed control. Both of these have extended the scope of industrial control applications significantly. Nowadays, real-time and distributed controls have become the governing control concepts and rationales of a vast range of systems, from medical instruments in hospitals to satellites in the sky.

This chapter discusses three important industrial control systems; embedded control systems, real-time control systems, and distributed control systems. This chapter goes through the principles and functions, the architectures and components, and the implementation techniques for each of the three control systems.

1.1 EMBEDDED CONTROL SYSTEMS

1.1.1 Definition and functions

All of us are familiar with personal computers (PCs), laptop computers, workstation computers, supercomputers and so on. All computers have a common feature; they contain both hardware and software, integrated and packaged into one device. In hardware, modern computers have microprocessors and programmable integrated circuits to perform computing functions. In software, computers have both machine-level instructions and advanced programming languages to produce layers of program structure composed of firmware, the operating system and application programs.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10001-4

In industry control systems are used in a large range of applications. For example, in factories, chemical reactors can be connected to a computer to monitor the volumes of solutions; in petrol stations, fuel pumps are connected to computers to display the volumes and prices of fuel taken; in supermarkets, cameras are connected to computers to monitor customers' activities. All of these examples are defined as computer control.

In computer control, the computers playing the role of controller are neither inside the controlled devices nor incorporated into the controlled systems. In this case, both the input and output ports are connected to the devices or systems to be controlled. Using these interfaces, computers can read and send digital data to and from the controlled device/system. With such communications, computers fulfil their control functions.

In the 1970s, modern computers with microprocessors and programmable integrated-circuits first became available, and industries started using them in some production processes. However, there were very few applications where the computers were not incorporated into the controlled system. In most cases, both hardware and software were built into the systems, becoming system components; in other words, they were embedded control systems.

The first recognizable embedded control system was the Autonetics D-17 guidance computer for the Minuteman II missile, delivered in 1966. This was the first built with integrated circuits for processor and memory. In 1978, the National Engineering Manufacturers Association of America released a standard for a programmable microcontrollers including single-board computers, numerical controllers and logic controllers able to perform control instructions. About 10 years later, in the late 1980s, microprocessor chipsets were produced, and embedded controls became widely applicable. As modern computer and electronic techniques developed in the following years, embedded control has become the most important industrial control technology available.

An embedded control system is a specially organized computer system designed for some dedicated control function or task. Its distinct feature is that its input/output system is not connected to an external computer; the microprocessor that controls the system is actually embedded in the input/output interface itself. Though the hardware differences between an embedded controller and a standard computer are obvious, the differences in software are also substantial. While most computers are based on operating systems requiring large memory size, such as the Windows or Linux operating systems, the typical embedded control system uses a smaller operating system, which has been developed to provide a simple and powerful graphical user interface (GUI).

Although in embedded control, the central processing unit (CPU) will mainly run independently of any supervisory controller, it is always linked to other parts of the controlled system to obtain digital status information. In this sense, embedded control can be consider as a subset of the overall data acquisition process.

In summary, an embedded control system refers to the computer hardware and software which are physically embedded within a large industrial process or production system. Its function is to maintain some property or relationship to other components of the system, in order to achieve the overall system objective. Embedded control systems are designed to perform a specific control task, rather than multiple tasks as a general-purpose computer does. Embedded control systems are not always separate devices or components, but are often integral to the controlled devices. The software designed for embedded control systems comprises embedded firmware, embedded operating systems, and special application programs. Compared with general computer software packages, embedded system software is rather smaller in size, and thus able to run with limited hardware resources, and without the

keyboard and screen needed for most applications. Usually, the embedded control software is stored in memory chips rather than on a hard disk. Due to these differences, embedded control systems have achieved wide applicability in industry world-wide.

1.1.2 Architectures and elements

The architecture of an embedded control system consists of hardware and software elements (Figure 1.1). The hardware architecture provides interconnection for all hardware components. The software architecture allocates sequences of programs to specific general-purpose microprocessors. The architecture configuration largely depends upon the ingenuity of the architect. One factor impacting performance is whether a given function of the controlled system is implemented as hardware or software. In an embedded control system, especially in a distributed embedded system, the hardware and software are divided into several groups that run multiple processes concurrently; we define each of these groups as a module.

(1) The hardware architecture of embedded control systems

Embedded control systems are generally developed on the customer's hardware, and often require high quality and reliability, and compatibility with real-time controls. Many use distributed architectures, on which a large number of processes are able to run concurrently. Such systems will commonly have several different types of general-purpose microprocessors, microcontrollers and application-specific integrated circuits (ASIC), all interconnected with communication links to work with embedded

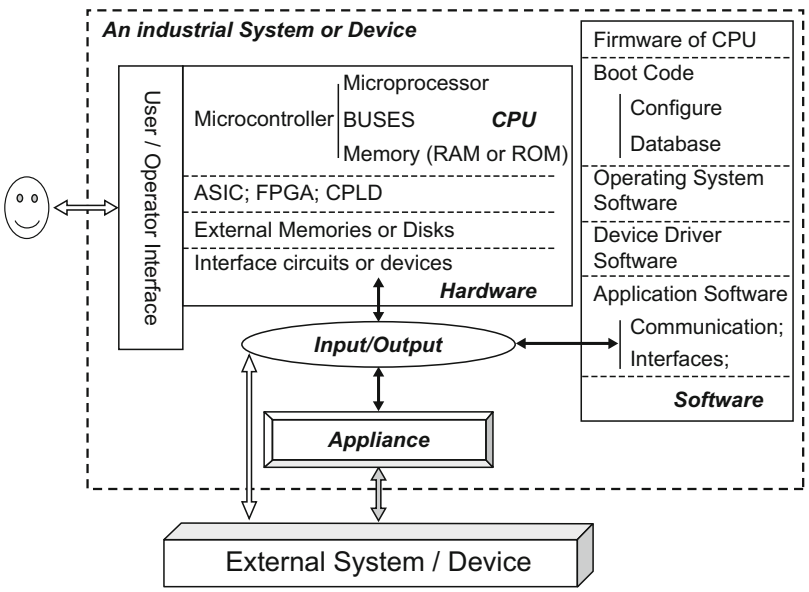


FIGURE 1.1

General architecture of an embedded control system.

software. It is crucial that each process can be executed with a variety of hardware modules and software programs.

The core of an embedded control system is a microprocessor, designed to operate a set of machine instructions, including firmware and boot-code, which are stored in the hardware memory. Another important hardware component is the microcontroller, typically a CPU core for the processing functions, together with a bus interface for communication with memory components and external or peripheral devices.

To provide flexibility, an embedded control system normally has some ASIC chips in addition to the microprocessor. An ASIC chip is designed for a specific application, rather than being a general-purpose CPU, since it does not incur the overhead of fetching and interpreting instructions stored in memory. ASIC chips can be based on standard cells, gate arrays, field-programmable gate arrays (FPGAs), or complex programmable logic devices (CPLDs). Embedded control systems employing reconfigurable hardware such as FPGA and CPLD are referred to as reconfigurable embedded systems. Reconfigurable systems can provide higher performance and flexibility to adapt to changing system needs at lower cost. Dynamically reconfigurable embedded control systems exploit the reconfiguration ability of programmable devices at run-time to obtain further cost saving.

An embedded control system should have storage elements; typically software executed by microprocessors or microcontrollers. For data storage, the microprocessor or microcontroller can include various types of memories, such as random access memory (RAM), read-only memory (ROM) or flash memory. Flash memory can be erased and reprogrammed in blocks instead of being programmed one byte at a time.

For transmitting and receiving data between devices and components, the microprocessor, microcontroller, ASIC and other devices may utilize various types of serial interfaces, parallel interfaces or buffers. Buffers are used by microprocessor to store operating data temporarily.

(2) The software architecture of embedded control systems

Many embedded control systems are required to fit real-time or multiple-process environments. Real-time or multiple-process operating systems are defined in the software, allowing the system to perform in a general-purpose control environment. Real-time operating systems, in comparison with non-real-time operating systems, offer much shorter response times. In multiple-process non-real-time operating systems, each process restarts the running of other processes of the same or lower priority level, and the response time can be much longer.

An embedded control system normally executes software in which the user interface is used as a system interface. A program package for application software is a necessary component of embedded control system software architecture. This application software runs under the control of a real-time operating system (RTOS) and is partitioned into one or more threads or tasks. A thread or task is an independent object of executable software that shares the control of the microprocessor with other defined threads or tasks within the embedded control system.

Embedded system software exists in many forms; such as software in a network router, or system software in a controlled device. Implementing a particular function in software provides flexibility, since it is more easily modified than hardware. Implementing a particular function in hardware, however, is generally performs faster. Hardware implementation may also reduce demand on microprocessors and, so speed up software execution.

The firmware and boot-code are also important parts of embedded system software, since they contain its configuration parameters. These allow an embedded control system to access and configure the system by setting its values. Different processes can run in different modules, which can work together if the configuration parameters of this system are properly set. In run-time, the behaviors of an individual module are governed by the values of the corresponding configuration parameters. The collection of configuration parameters from the whole system comprises a current configuration database, which determines the overall behavior of the entire system. This configuration database is included in the firmware and boot-code, so to improve system performance it is just necessary to upgrade the firmware and boot-code, or change this configuration database.

In distributed embedded control systems, communication protocols are encoded in software. The software that is responsible for communications between different components is sometimes called communication software.

Most of the software of embedded control systems is a special package working as system routines. These provide information to the user about how to use, configure, diagnose, and trouble-shoot a device. Some embedded control systems also contain calibration routines in software and hardware to check and correct system working parameters.

1.1.3 Implementation methods

The implementation of embedded control in any industrial system or device will involve multi-disciplinary engineers, and include many types of technical projects. Figure 1.2 explains this topic, from which we learn that the completion of an embedded control system includes:

1. planning automatic control strategies;
2. verifying machine systems and components;
3. engineering of embedded software including architecture design, code programming and testing;
4. engineering of embedded hardware including computer components and electronics components;
5. carrying out dependability tests and evaluations.

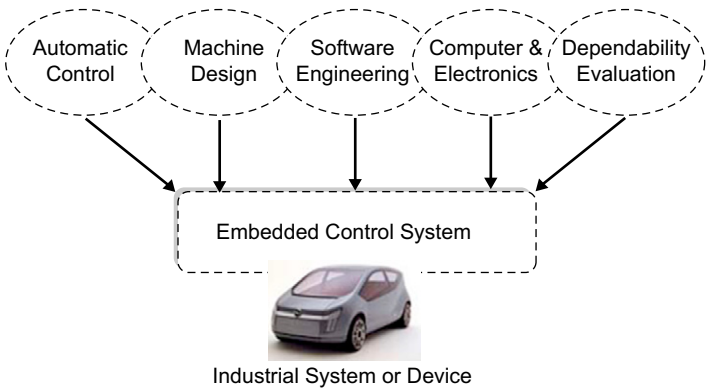


FIGURE 1.2

The implementation of an embedded control system in industry involves disciplinary engineering.

Since the twentieth century, effective methodologies have been developed for implementing embedded control systems in industry. This textbook introduces two of them; control function mapping for implementing hardware of embedded control systems; and control logic modeling for implementing the software of embedded control systems.

(1) Control function mapping for hardware implementation

In the following explanation, the controlled system or device is considered as being divided into two subsystems or components (Figure 1.3).The method starts by providing a data map of the controlled system or device. The first data map should contain several first data map points. Each of the first data map points represents a first data map output value. Then, a function map is developed which comprises a second data map of the controlled system or device. The second data map should contain a number of second data map points. Each of the second data map points corresponds to one of the first data map points. The second data map is divided into a first-type data map region containing second data map points representing only second data map output values from the first-type, and a second-type data map region. This second-type data region contains second data map points representing second data map output values only of the second type, whereby a portion of the second data map defines a hysteresis region.

Subsequently the method determines an operating point on an operating path within the second data map in dependence upon the first and the second operating parameters of one subsystem or component of the controlled system or device. Then it determines a control function for another subsystem or component based on a first data map output value determined from the first data map and

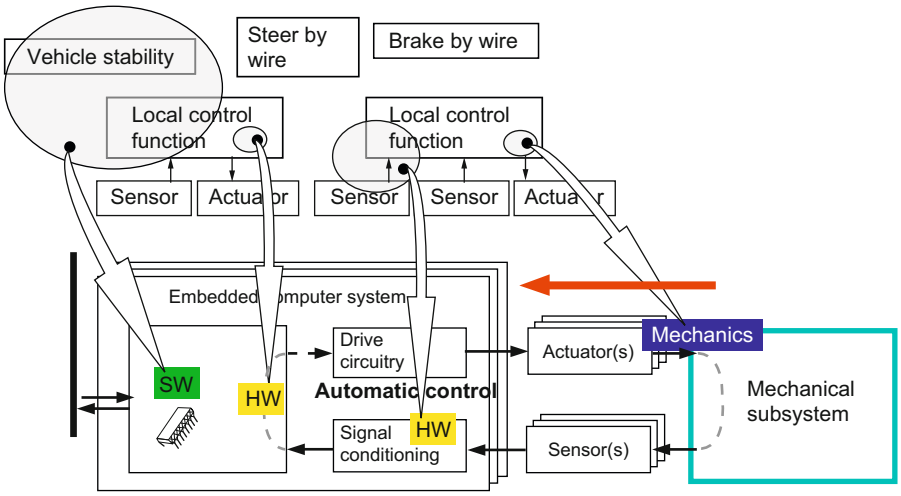


FIGURE 1.3

An explanation of control function mapping for implementing embedded control system hardware in a car. This car is assumed to have two controlled components: steer and brake.

the second data map output value determined from the second data map, in accordance with the two criteria:

(a) If the operating point in the second data map locates in a part of the first-type data map region which is outside the hysteresis region, the second data map output value should be an output from the first-type data map region. The first data map output value is interpolated from first data map output values of the first data map points adjacent to or neighboring the first data map point corresponding to the operating point in the second data map.

(b) Or, if the operating point in the second data map locates in a part of the first-type data map region which is within the hysteresis region, a check is made to deteremine whether the operating point in the second data map entered the hysteresis region from a previous operating point on the operating path within the first-type data map region. Then the second data map output value should be output from the first-type data map region and a first data map output value which is interpolated from the first data map output values of the first data map points adjacent to or neighboring the first data map point corresponding to the operating point in the second data map.

(2) Control logic modeling for software implementation

After control strategies are derived, the next task is the implementation of the hardware and software of the embedded control system. Both control theories and control strategies are topics beyond the scope of this textbook, so the following text will focus on implementing software using the control logic modeling method.

First, the software architecture for the system must be chosen. There are two types of software architectures available; the layers model and the classes model. In an embedded control system, the layers model divides the whole software organization into six layers according to the execution sequence of commands sent from the user interface. The part of the interface which first receives a command is defined as high level; the CPU located in the microprocessor chipset which finally fulfils this command is defined as low level. Please note that the layer definition here differs from the protocol for computer networks as given in the OSI (Open System Interconnect) network communication standard model. As shown in Figure 1.4(a), the six layers considered for embedded control system architecture are:

(a) User interface layer. This layer is responsible for the communication and interaction between the user and the controlled system. It receives user instructions and passes them down to a lower layer, then displays the returning results from the system to the user. This layer normally contains graphic objects such as buttons, bars, and dialog boxes, etc.

(b) Process handling layer. This layer is responsible for creating, managing and eliminating a process (also referred to as a task). It creates a process whenever it receives user instruction, then handles the running of this process, including getting parameters, managing concurrency and mutual exclusivity, etc. When a process is completed, it is deleted.

(c) Component control layer. Here, component is defined as one subsystem or individual part of the controlled system or device. For example, the scaning subsystem is one component of a copying machine; the print subsystem is another. One system or device can have several components for its control. The responsibilities of this layer include communication and interaction with the higher and lower layers, and the managing and handling of the processes running within this component.

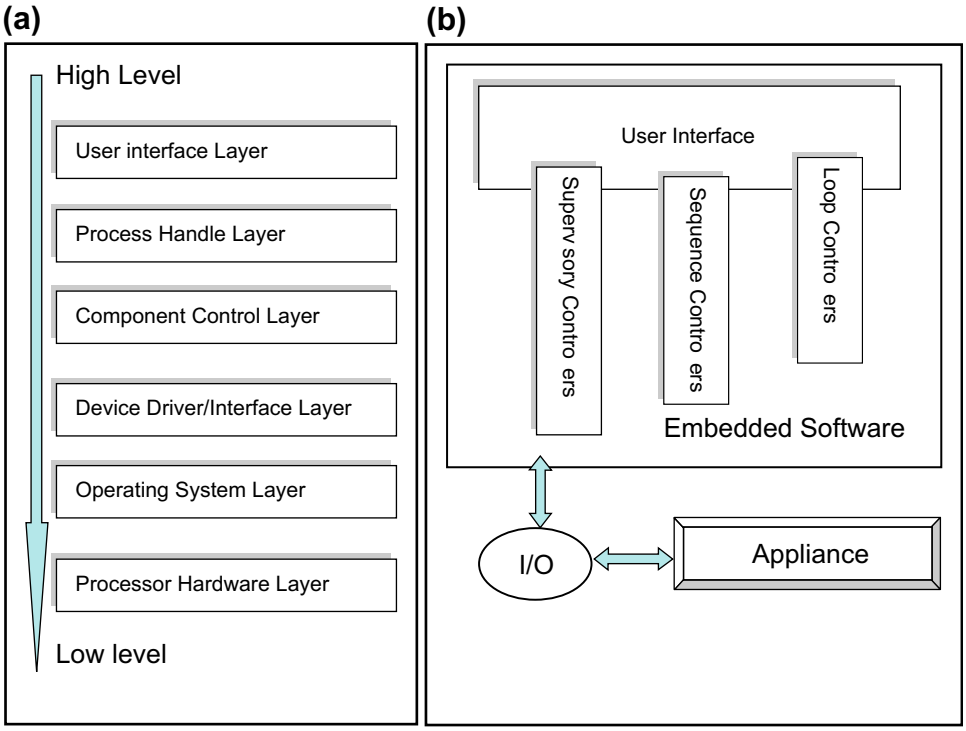


FIGURE 1.4

Two types of software architectures for embedded control systems: the left panel (a) is the layers model; the right (b) is the classes model.

(d) Device driver/interface layer. In this context device means a subsystem or a part of the component defined above. For example, the scanning component of a copying machine can have such a device as its paper transport driving circuit, its digital image reading circuits and the circuits that interface with other components or devices in the machine. Because it mostly consists of firmware hardcoded in programmable integrated circuits or in ASICs, this layer can be included in the operating system layer.

(e) Operating system layer. The same as a computer's operating system software, this layer is the shell between the microprocessor chipset and the outer components of the controlled system. It schedules processes (or tasks), manages resources such as memory and disks, and handles communication with peripherals and components.

(f) Processor hardware layer. This layer comprises the firmware of the CPU and the boot-code in the microprocessor chipset.

In the classes model of software architecture, embedded control systems are divided into three parts; embedded software, I/O interfaces and appliances. As shown in Figure 1.4(b), the embedded software is separated from the I/O interface and appliance. The example in Figure 1.4(b) contains three

controllers and the user interface. The three controllers included in embedded software are: the loop controllers which implement the control strategies, and fulfil the real-time control standards; the sequence controllers, which implement sequences of control events based on control logics, and also manage the loop controllers; and the supervisory controllers which issue expert systems optimization algorithms that adapt parameters of the lower two controllers. The user interface communicates with these controllers and interacts with users. The dynamic behaviour of the controlled system is essential for the embedded control functionality. The I/O interface chipsets, also separated from embedded software, are dedicated to the embedded control system.

After selecting the software architecture, implementation proceeds to the program design, critical for effective embedded software. The program design comprises these tasks:

(a) Splitting control strategies into the logic of the sequence controllers, the loop controllers and the supervisory controllers. The programmed objects of each of these is then designed and implement then

(b) Assigning every component or device inside the controlled system, including all the I/O interfaces, to a corresponding programmed object; filling each of these program objects with the necessary sub-objects based on the subsystems and the devices present.

(c) Incorporating external instructions from the users and the commands from connected systems into the executing sequences of these programmed objects.

The following tools will be very useful for efficient design. The first is a data flow diagram for the software structure. In such a diagram, the vertices denote the processes and the edges denote the data communications between program objects.

The second tool is VHDL, which is an acronym for "very high-speed integrated circuits (VHSIC) hardware description language". VHDL is used to develop ASICs and FPGAs. The third tool is a bond graph. Bond graphs are direct graphs showing the dynamic behavior of the controlled systems. Here, vertices are the sub-models, and edges denote the ideal exchange of energy. They are physical-domain independent, due to analogies between these domains on the level of physics. This has the advantage that mechanical, electrical, hydraulic, light, etc. system parts can all be modeled with the same graphs.

The next steps in implementing the embedded software are coding and testing the software. For these two topics, please refer to the Part Eight of this textbook, which provides detailed discussions on this subject.

1.2 REAL-TIME CONTROL SYSTEMS

1.2.1 Definition and functions

There are several different definitions of a real-time control system in the literature and documents concerning control and automation. In some industrial markets, vendors and engineers claim that they have made real-time instruments without, however, defining what they mean by the term. For example, some washing machines claim to have real-time double cylinder drivers.

Real-time control should be one criterion of control operations to be fulfilled by industrial control systems. A control operation is a series of events or actions occurring within system hardware and software to give a specific result. A real-time control system is a system in which the correctness of

a result depends not only on its logical correctness but also on the time interval in which the result is made available. This is to say that time is one of the most important aspects of control systems. In general, the following three standards give the definition of a real-time control operation, and an industrial control system in which all the control operations occur in real-time qualifies as a real-time control system:

(1) Reliable operation execution

By reliable operation execution we mean two things: the operation execution must be stable, and it must be repeatable. Because a control operation is executed with both the system hardware and software, stable and repeatable operations represent the capability of the hardware, the software, and their mutual compatibility in this respect. Stable operation requires that the system must produce exactly the same result for any given set of input data or events within the same time frame. If a control operation provides different results with the same inputs, this operation is unstable; if a control operation produces the same result for the same inputs but exhibits variability in the time frame it takes from input to output, this operation cannot be considered to be real-time. Obviously, a precondition of stable operations is that this control system must support repeatable execution of any control operation with the same hardware and software environments.

(2) Determined operation deadline

Any control operation needs time to execute. At the instant of getting an input or recognizing an event, the system must start an operation; at the instant this operation is completed, the system must generate an output or give a response. As mentioned earlier, a key factor to any real-time control system is the time interval an operation takes between a new input or a new event, and the instant the response or update is given. Therefore, the real-time criterion requires that this time interval must be determined, and be within a confirmed deadline for executing any control operation.

(3) Predictable operation result

Finally, the result for any control operation must be predictable. This means there is no uncertainty which could affect the result of any control operation. For a real-time control system, it is not acceptable for the result of the control operation to be dependent upon factors that are unknown. Any factors that have an effect on the output state, for a given input state, must be known and controllable.

Although a real-time control operation should satisfy these three criteria, it is essential to identify a time frame for real-time operation because this factor is crucial in an industrial control system environment. For example, missing the deadline of critical servo-level periodic tasks could result in losing data or crashing control cycles, which could also lead to some loss of efficiency at best, or at worst could cause serious damage to equipment or cause human injury. However, it should be noted that real-time control is not the same as rapid control. Fast control aims at getting the results as quickly as possible, while real-time control aims at getting the results at a determined point of time within defined time tolerances, or deadlines.

These systems originated in the early twentieth century, with the need to solve two main types of control problem; event response, and closed-loop control systems. Event response requires that a system should respond to a stimulus in a determined amount of time, for instance in operations an automotive airbag system. Closed-loop control systems process feedback continuously in order to adjust an output; an automotive cruise control system is an example of a closed-loop control system. Both of these types of

systems require the completion of an operation within a specific deadline. This type of performance is referred to as determinism.

To understand a real-time control system, we need to clarify several relevant terms:

The ready time of a task is the time instant when the task becomes available to the system, which means the system has built up the task object in the memory of the microprocessor chipset and inserted this object into the operating system kernel's task queue.

The start time is the earliest time the task can start executing (always greater than or equal to the ready time).

The completion time is the time instant at which the task normally finishes running.

The execution time of a task is the time interval taken by the system for the whole processing of this task between the start time and the completion time.

The response time is the interval between the ready time and the completion of the execution.

The deadline is the time when the task's execution must be stopped.

If the completion time of a task is not earlier than the deadline, the task is late.

All real-time systems have a certain level of jitter. Jitter is a variance in the actual timing of the above times. In a real-time system, jitter should be within a given interval so that system performance can still be guaranteed.

Figure 1.5 is an illustration of these temporal task terminologies.

In their application, real-time control systems are divided into two classes; soft and hard. These two types of real-time control are distinguished in terms of the deadline of control operations.

In hard real-time systems, only those operations which can meet the prescribed deadlines are valid; the completion of an operation after its deadline is considered useless, and if the operations keep on missing the prescribed deadlines, the entire system will eventually fail. Hard real-time systems are typically supported by physical hardware in embedded control when it is imperative that an event is reacted to within a strict deadline. For example, nuclear power stations and car airbags must turn-on or turn-off immediately, when the corresponding commands arrive, otherwise they will physically damage their surroundings or threaten as human life. Another example is medical instruments such as heart pacemakers. A heart pacemaker is a battery-powered electronic device about the size of a matchbox which produces electrical current in order to cause the heart to beat in a regular and reliable manner. The delayed operation of a heart pacemaker could result in loss of human life.

Soft real-time means that the control system can tolerate some deadline missing with only decreased service quality or system malfunction resulting. Thus, soft real-time systems are typically

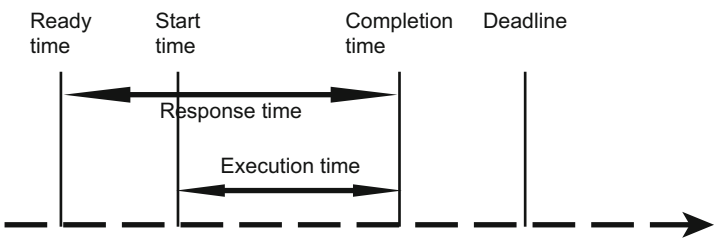


FIGURE 1.5

Several terminologies relevant to task (process, job) temporal measures.

those used where there is some issue of concurrent access, and there is a need to keep a number of connected systems up to date with changing situations. For example, when pressing a cell phone button to answer an incoming call, the connection must be established soon after the button has been pressed, but the deadline is not mission-critical and small delays can be tolerated.

In conclusion, by a real-time control operation we mean those tasks or processes running in computerised industrial control systems or devices which satisfy three criteria: reliable operation execution, set operation deadline, and predictable operation results. The time deadline is crucial for real-time control operations. If the operation must meet the deadline, otherwise risking fatal errors or undesirable consequences, it is called a hard real-time operation. In contrast, if the meeting of the deadline for this operation is desirable but not essential, it is called a soft real-time operation. All computerized industrial control systems which provide real-time operations are defined as real-time control systems.

1.2.2 Architectures and elements

Time is an important factor that should be taken into account in an industrial environment, and is key to the wide adoption of real-time control systems in industrial plants and workshops. Real-time control systems are defined as those systems in which the correctness of the system depends not only on the logical result of controls, but also on the time at which the results are produced. A real-time control system in industry requires both dedicated hardware and software to implement real-time algorithms. Many explanations of the architecture of real-time control systems used reference model terminologies; these seen unnecessary for us to understand the architecture of real-time control systems. Advanced real-time control systems are based on real-time algorithms. This textbook will focus on advanced computer-based real-time control systems which require the following hardware and software to operate real-time controls.

(1) Real-time control system hardware

Real-time hardware includes both microprocessor unit chipsets and programmable peripherals. Microprocessor units are necessary because they are the core of modern digital computers and perform the actual calculations necessary; programmable peripheral interfaces are also necessary they are important auxiliary hardware of modern digital computers and comprise the means of recieving and communicating data.

(a) Microprocessor units

A microprocessor unit is the unified semiconductor of electronic integrated circuits which performs the central processing functions of a computer. A microprocessor unit comprises a CPU (central processing unit), buffer memories, an interrupt control unit, input/output interface units and an internal bus system. Chapter 5 of this textbook discusses this topic in detail.

(b) Programmable peripheral devices (memory chips)

Programmable peripheral interfaces consist of several programmable integrated-circuit units attached to microprocessor units to allow the microprocessor to perform interrupt controlling, timer controlling, and to also control dynamic memory volumes and input/output ports. In industrial controls, controllers need a large quantity of their function information about the systems to perform. It is important that

real-time controllers request information dynamically about the controlled systems as resources. To store the information dynamically on controlled systems, such microprocessors must expand their memories using dedicated peripherals such as direct memory access (DMA), non-volatile memory (NVM), etc. Chapter 6 of this textbook discusses this topic in detail.

(c) Industrial computers and motherboards

Nowadays, most industrial computers are able to perform real-time controls because they have necessary microprocessor units and peripherals. An industrial motherboard is a single-board computer normally of a single type. Both industrial computers and motherboards will be explained in Chapter 9 of this textbook.

(d) Real-time controllers

Some dedicated controllers are to real-time control. The PLC (programmable logic control) controller is a typical example. Other examples of real-time controllers are CNC (computer numerical control) controllers, PID (proportional-integral-derivative) controllers, servo controllers and batch controllers. Most modern SCADA (supervisory control access and data acquisition) control systems also contain real-time controllers. In this textbook, Part 4 Chapters (7 and 8) discuss most types of real-time controllers and robots.

(2) Real-time control system software

As mentioned earlier, a real-time control system should be a computerized system that supports real-time control algorithms in its hardware and software. It seems that there is no single definition of a real-time algorithm, and what a real-time algorithm includes. Many have been developed such based on own control applications; thus different real-time algorithms are adapted to different varying systems.

In a, the operating system is responsible for performing these algorithms. An operating system that can handle real-time algorithms is known as a real-time operating system (RTOS). Although Chapter 16 of this textbook is devoted to this topic, a briefer explanation is provided below.

(a) Multitask scheduling

There is a task dispatcher and a task scheduler in a RTOS, both of which handle multi-task scheduling. The dispatcher is responsible for the context switch of the current task to a new task. Once a new task arrives (by means of the interrupt handling mechanism), the context switch function first saves the status and parameters of the current task into memory then, and it terminates; it then transfers to the scheduler to select a task from the system task queue. The selected task is then loaded and run by the system CPU.

The scheduler takes the job of selecting the task from the system task queue. Two factors determine the difference between a RTOS and a non-RTOS; the first is the algorithm which select this task from the system queue, and the second is the algorithm that deals with multitasking concurrency.

There are two types of algorithms for task selection: static and dynamic scheduling. Static scheduling requires that complete information about the system task queue is available before the scheduler starts working. This includes the number of tasks, its deadline, its priority, and whether it is a periodic or aperiodic task, etc. Obviously, the static algorithm must always be planned when system is off-line. The advantage of off-line scheduling is its determinism, and its disadvantage is its

inflexibility. In contrast, dynamic scheduling can be planned when the system is either off-line or on-line. Dynamic scheduling can change its plan as the system status changes. The advantage of dynamic scheduling is its flexibility, and its disadvantage is its poor determinism.

The term "concurrency" in a RTOS means several tasks executing in parallel at the same time. Concurrency supported is a broad range of microprocessor systems, from tightly coupled and largely synchronous parallel in systems, to loosely coupled and largely asynchronous distributed systems. The chief difficulty of concurrency implementation is the race conditions for sharing the system resources, which it can lead to problems such as deadlock and starvation. A mutual exclusion algorithm is generally used to solve the problems arising from race conditions.

An RTOS must guarantee that all deadlines are met. If any deadline is not met, the system suffers overload. Two algorithms are used to meet the deadlines in the RTOS; either pre-emptive-based or priority-based algorithms.

At any specific time, tasks can be grouped into two types: those waiting for the input or output resources such as memory and disks (called I/O bound tasks), and those fully utilizing the CPU (CPU bound tasks). In a non-RTOS, tasks would often poll, or be busy/waiting while waiting for requested input and output resources. However, in a RTOS, these I/O bound tasks still have complete control of the CPU. With the implementation of interrupts and pre-emptive multitasking, these I/O bound tasks could be blocked, or put on hold, pending the arrival of the necessary data, allowing other tasks to utilize the CPU. As the arrival of the requested data would generate an interrupt, blocked tasks could be guaranteed a timely return to execution.

The period for which a microprocessor is allowed to run in a pre-emptive multitasking system is generally called a time slice. The scheduler is run once for each time slice to choose the next task to run. If it is too short, then the scheduler will consume too much processing time, but if it is too long then tasks may not be able to respond to external events quickly enough. An interrupt is scheduled to allow the operating system kernel to switch between tasks when their time slices expire, allowing the microprocessor time to be shared effectively between a number of tasks, giving the illusion that it is dealing with these tasks simultaneously. Pre-emptive multitasking allows the computer system to guarantee more reliably each process a regular slice of operating time. It also allows the system to deal rapidly with external events such as incoming data, which might require the immediate attention of one task or another. Pre-emptive multitasking also involves the use of an interrupt mechanism, which suspends the currently executing task and invokes a scheduler to determine which task should execute next. Therefore all tasks will get some amount of CPU time over any given period of time.

In the RTOS, the priority can be assigned to a task based on either the task's pre-run-time or its deadline.

In the approach that uses pre-run-time-based priority, each task has a fixed, static priority which is computed pre-run-time. The tasks in the CPU task queue are then executed in the order determined by their priorities. If all tasks are periodic, a simple priority assignment can be done according to the rule; the shorter the period, the higher the priority.

There are some dynamic scheduling algorithms which assign priorities based on the tasks' respective deadlines. The simplest of this type is the earliest deadline first rule, where the task with the earliest (shortest) deadline has the highest priority. Thus, the resulting priorities of tasks are naturally dynamic. This algorithm can be used for both dynamic and static scheduling. However, absolute deadlines are normally computed at run-time, and hence the algorithm is presented as dynamic. In this approach, if all

tasks are periodic and pre-emptive, then it is optimal. A disadvantage of this algorithm is that the execution time is not taken into account in the assignments of priority.

The third approach in assigning priorities to the tasks is the maximum laxity first algorithm. This algorithm is for scheduling pre-emptive tasks on single-microprocessor systems. At any given time, the laxity (or slack) of a task with a deadline given is equal to:

$$\text{Laxity} = \text{deadline} - \text{remaining execution time}.$$

This algorithm assigns priority based on their laxities: the smaller the laxity, the higher the priority. It needs to know the current execution time, and the laxity is essentially a measure of the flexibility available for scheduling a task. Thus, this algorithm takes into consideration the execution time of a task, and this is its advantage over the earliest deadline first rule.

Every commercial RTOS employs a priority-based pre-emptive scheduler. This is despite the fact that real-time systems vary in their requirements and real-time scheduling is not always needed. Multitasking and meeting deadlines is certainly not a one-size-fits-all problem. Figure 1.6 presents a classification for the most well-known dynamic scheduling algorithms for single microprocessor systems.

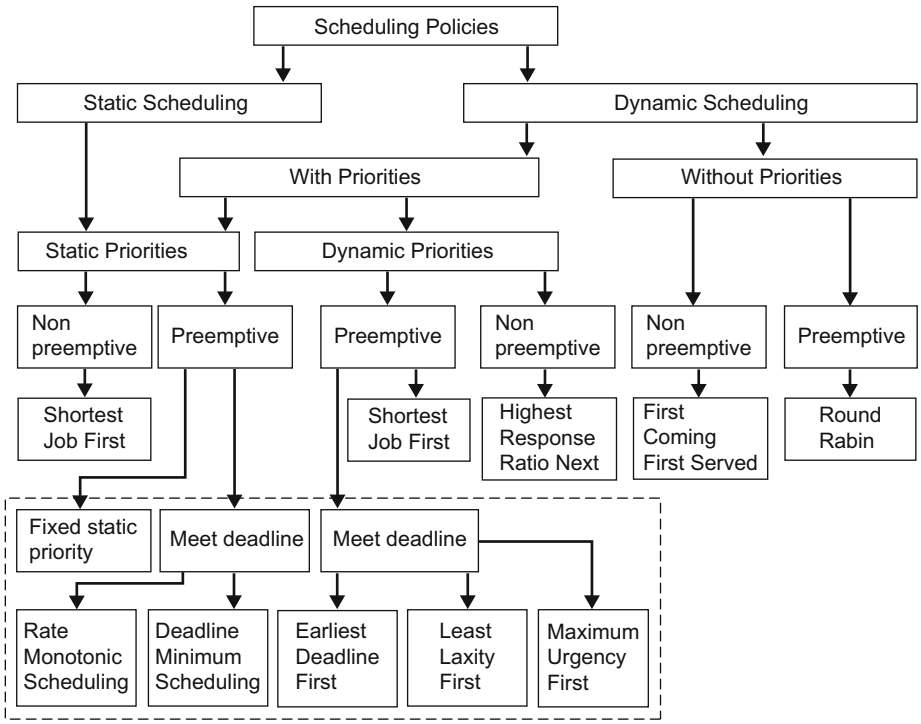


FIGURE 1.6

A classification for the most well-known multitasking scheduling algorithms for single microprocessor systems. Please note that only those algorithms encompassed by the dashed box are real-time multitasking scheduling algorithms.

(b) Intertask communications

An RTOS requires intertask communication, as system resources such as data, interfaces, hardware and memories must be shared by multiple tasks. When more than one task concurrently requests the same resource, synchronization is required to coordinate these requests. Synchronization can be carried out in a RTOS by using various message queue program objects.

Two or more tasks can exchange information via their common message queue. A task can send (or put) a data frame called a message into this queue; this message will be kept and blocked until the right task accesses it. In a RTOS, the message queue can be of the mailbox, pipe or shared memory type.

The system common message queue is also a critical section which cannot be accessed by more than one task at a time. Mutually exclusive logic is used to synchronize tasks requesting access to this queue. All types of system message queue use mutual exclusion, including busy-waiting, semaphore and blocked task queue protocols, to synchronize the requested accesses to the system message queue.

(c) Interrupt handling

It can be said that the interrupt handler is a key function of a RTOS. Usually, the RTOS allocates this interrupt handling responsibility to its kernel (or nucleus).

General-purpose operating systems do not usually allow user programs to disable interrupts, because the user program could then control the CPU for as long as it wished. Modern CPUs make the interrupt disable control bit (or instruction) inaccessible in user mode, in order to allow operating systems to prevent user tasks from doing this. Many embedded systems and RTOS allow the user program itself to run in kernel mode for greater system call efficiency, and also to permit the application to have greater control of the operating environment without requiring RTOS intervention.

A typical way for the RTOS to do this is to acknowledge or disable the interrupt, so that it would not occur again when the interrupt handler returns. The interrupt handler then queues work to be done at a lower priority level. When needing to start the next task, the interrupt handler often unblocks a driver task through releasing a semaphore or sending a message to the system common message queue.

(d) Resource sharing

In a real-time system, all the system resources should be shared by system multitasks, and so they are called shared resources. Thus, algorithms must exist for the RTOS to manage sharing data and hardware resources among multiple tasks because it is dangerous for more than two tasks to access the same specific data or hardware resource simultaneously. There are three common approaches to resolve this problem: temporarily masking (or disabling) interrupts; binary semaphores; and message passing.

In temporarily masking (or disabling) interrupts, if the task can run in system mode and can mask (disable) interrupts, this is often the best (lowest overhead) solution to preventing simultaneous access to a shared resource. While interrupts are masked, the current task has exclusive use of the CPU; no other task or interrupt can take control, so the shared resource is effectively protected. When the task releases its shared resources, it must unmask interrupts. Any pending interrupts will then execute. Temporarily masking interrupts should only be done when the longest path through the shared resource is shorter than the desired maximum interrupt latency, or else this method will increase the system's maximum interrupt latency. Typically this method of protection is used only when the shared resource is just a few source code lines long and contains no loops. This method is ideal for protecting hardware bitmapped registers when the bits are controlled by different tasks.

When the accessing resource involves lengthy looping, the RTOS must resort to using mechanisms such as semaphores and intertask message-passing. Such mechanisms involve system calls, and usually on exit they invoke the dispatcher code, so they can take many hundreds of CPU instructions to execute; while masking interrupts may take very few instructions on some microprocessors. But for longer resources, there may be no choice; interrupts cannot be masked for long periods without increasing the system's interrupt latency.

In run-time, a binary semaphore is either locked or unlocked. When it is locked, a queue of tasks can wait for the semaphore. Typically a task can set a timeout on its wait for a semaphore. Problems with semaphore are well known; they are priority inversion and deadlocks. In priority inversion, a high-priority task waits because a low-priority task has a semaphore. A typical solution for this problem is to have the task that has a semaphore run at (inherit) the priority of the highest waiting task.

In a deadlock, two or more tasks lock a number of binary semaphores and then wait forever (no timeout) for other binary semaphores, creating a cyclic dependency graph. The simplest deadlock scenario occurs when two tasks lock two semaphores in lockstep, but in the opposite order. Deadlock is usually prevented by careful design, or by having floored semaphores. A floored semaphore can free a deadlock by passing the control of the semaphore to a higher (or the highest) priority task predefined in a RTOS when some conditions for deadlock are satisfied.

The other approach to resource sharing is message-passing. In this paradigm, the resource is managed directly by only one task; when another task wants to interrogate or manipulate the resource, it sends a message, in the system message queue, to the managing task. This paradigm suffers from problems similar to those with binary semaphores: priority inversion occurs when a task is working on a low-priority message, and ignores a higher-priority message or a message originating indirectly from a high-priority task in its in-box. Protocol deadlocks occur when two or more tasks wait for each other to send response messages.

1.2.3 Implementation methods

In general, building a real-time control system involves two stages. The first stage is the design of the control laws or models. The second stage is the implementation of the system in hardware and software. It seems that most modern control systems are implemented digitally. We will therefore focus on the digital implementation of real-time control systems in this section.

Figure 1.7 shows the general form of a digital real-time control system in which the *Computer* can be also a digital controller; u(k) and u(t) denote the digital and analog control signals, respectively; r(k) and r(t) denote the digital and analog reference signals, respectively; y(k) and y(t) denote the digital and analog process outputs, respectively. From Figure 1.7, we can see that the digital control system contains not only wired components but also algorithms that must be programmed into software. The majority of control laws applied to digital control systems can be represented by the general equation:

$$P(q^{-1})\,u(k) = T(q^{-1})\,r(k) - Q(q^{-1})\,y(k) \tag{1.1}$$

where P, T, and Q are polynomials.

In real-time implementation, timing is the most important issue. In reality, delays in timing are present in both the signal conversion hardware and the microprocessor hardware. If the delay is of fixed duration, then Equation (1.1) can often be modified to compensate for it. Unfortunately, the delays in a digital control system are not usually fixed. The following are two examples of this

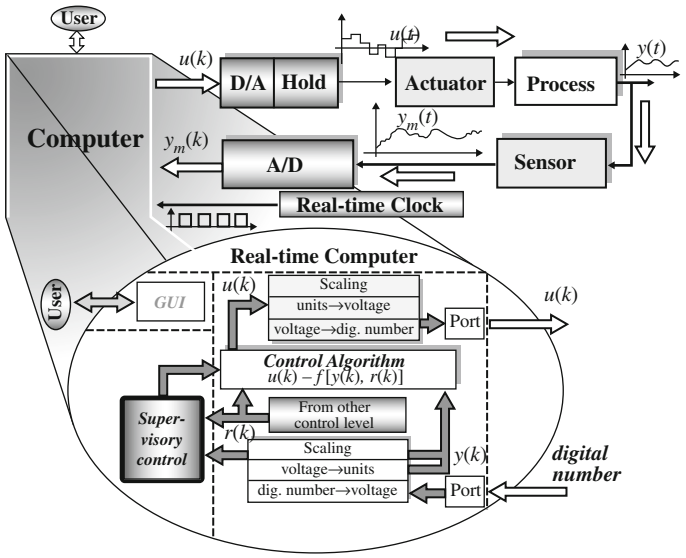


FIGURE 1.7

General configuration of digital real-time control systems.

problem. Some microprocessors, especially those with complex instruction-set architecture such as the Pentium, can execute a varying number of CPU cycles. This generates a non-fixed delay duration. In additional, modern microprocessors serve interrupts based on the priority of each individual interrupt. If the CPU is serving a lower-priority interrupt, and a higher-priority interrupt arrives, the CPU must stop and immediately transfer to serve the new interrupt. This also results in a non-fixed delay.

At the implementation stage, all tasks should be scheduled to run with the available microprocessors and available hardware resources. The sampling time should be decided on the limited computation time provided by the system hardware. Thus, the computation time delay (control latency), τ is always in conflict with the sampling time, Ts. Depending on the magnitude of τ relative to Ts, the conflict can be either a delay ($0 < \tau < Ts$) or a loss ($Ts >= \tau$) problem. Because the time delay (control latency) originates from control errors, delay and loss can occur alternately in the same system at different times. If the control signal at time k is denoted by u(k), when the controller or computer fails to update its output during any one sampling time interval, u(k−1) should be applied again to test; the loss of the control signal occurs. Because this could occur randomly at any time, the failure to deliver a control signal can be treated as a correlated random disturbance Δu(k) at the input of the control system.

In real-time control systems, there is close linkage between control performance and the multitasking scheduling. Hence, the control system design has to be re-evaluated in order to introduce real-time control considerations, which could modify the selected task period, attributes etc. based on the control performance.

The following is an example of one-step-ahead predictive controller. It supposes that, in order to deliver the control signal u(k) as soon as possible, the one-step-ahead predictive controller can be used in the form:

$$u(k+1) = f[r(k+1), r(k), \ldots, r(k-n), u(k), \ldots, u(k-m), \hat{y}(k+1), y(k), \ldots, y(k-n)] \tag{1.2}$$

where f denotes a control law, u is the control signal, r the reference signal and y is the output signal of the control system. If r(k+1) is assumed to be known, y^(k+1) is calculated by a simple linear predictor given by:

$$\frac{\hat{y}(k+1) - y(k)}{(k+1) - k} = \frac{y(k) - y(k-1)}{k - (k-1)} \tag{1.3}$$

Or:

$$\hat{y}(k+1) = 2y(k) - y(k-1) \tag{1.4}$$

Hence, the control task started by first delivering u(k) which was calculated at time k−1. When this is done, the remaining operations are; reading the y(k) from the analog-to-digital signal converter, calculating y^(k+1), and then u(k+1). In this example, there is another approach which is based on the implementation of two periodic tasks. The first step calculates the control signal directly after reading and conditioning y(k), and the second step updates the states after the control signal value is delivered.

(1) Hardware devices

There are a variety of well-designed real-time hardware platforms in the industrial control market for implementing real-time control systems. The following are some generic examples.

(a) Microprocessor chipset

A microprocessor chipset consists of several, specially designed, integrated circuits which incorporate a digital signal processing function and a data acquisition function. There are two types of microprocessor chipsets available for developing real-time systems. The first of these is the generic microprocessor, which is either a multipurpose digital signal processing device or other, general microprocessor. The second is the embedded microprocessor, which can be incorporated into products such as cars, washing machines, industrial equipment, and so on to play a real-time digital signal processing role. A microprocessor chipset can have a CPU, memory, internal address and data bus circuits, registers, interfaces for peripherals, BIOS (basic input/output system), instruction set and timer clock, etc.

Microprocessors are characterized by the technical parameters: bandwidth, which is the number of bits processed in a single instruction (16, 32, and 64 bits nowadays); clock speed, which gives the number of instructions executed per second in units megahertz (MHz); memory volume, which is the storage capacity of the microprocessor memory in units of megabytes (Mb); instruction sets, which are the built-in machine-language instructions defining the basic operations of the microprocessor.

There are several large corporations making such microprocessor chipsets: Intel, AMD, Motorola, IBM, are examples of companies that have been producing microprocessors for different applications for a number of years following.

(b) Single-board computers

Single-board computers are complete computers with all components built on to a single board. Most single-board computers perform all the functions of a standard PC, but all of their components single or dual microprocessor, read-only memory (ROM) and input/output interfaces - must be configured on a single board.

Single-board computers are made in two distinct architectures: slot-supported and no-slot. Slot-supported single-board computers contain from one to several slots, in which one or some system buses connect with a slot processor board or multiple-slot backplane. There can be up to 20 slots on one backplane. This slot-supported architecture makes rapid system repair or upgrade by board substitution possible, and gives considerable scope for system expansion. Most single-board computers are made in a rack-mount format with full-size plug-in cards, and are intended to be used in a backplane. Selection of single-board computers should be application-oriented. Single-boards can be used either for industrial process control or in a plug-in system backplane as an I/O interface.

In addition to the big manufacturers such as Intel and AMD, there are many companies around the world that make different types of single-board computers. These companies have produced many products such as the CompactPCI, PCI, PXI, VXI, and PICMG, etc. These single-board computers are characterized by their components: CPU, chipset, timer, cache, BIOS, system memory, system buses, I/O data ports, interfaces and the slots that are generally standard PCI slot card.

(c) Industrial computers

In additional to single-board computers, other types of industrial computers are used in implementing real-time control systems.

A desktop PC can be used for real-time control. However, because standard operating systems do not use hard real-time semantics, such systems cannot be used for hard real-time controls. Desktop PC can be used as supervision and surveillance tools in real-time control systems. A typical example is SCADA systems, where desktop PCs are be used as supervisory and surveillance stations.

To work as real-time computers, all industrial computers real-time embedded software and compatible hardware. Converting an industrial computer into a real-time computer is not always a simple project, so single-board computers are popular in real-time control systems due to their flexibility.

(d) Real-time controllers

Real-time digital controllers are special-purpose industrial devices used to implement real-time control systems. There are two types; the first are called real-time controllers, and are specially made industrial computers that are adapted for industrial process and production controls. The second are purpose-built digital controllers such as PLC, CNC, PID, and fuzzy-logic controllers which are embedded with a RTOS in their own microprocessor units. The first type was discuss in the section on hardware. The second kind of digital controllers will be discussed in Part 4 of this textbook.

(2) Real-time programming

Embedded software architecture is a package of software modules for building control systems which are embedded within real-time devices. In this textbook, section 1.2.3 and Figure 1.4 discuss the software of embedded control systems in detail. In reality, most real-time control systems are

embedded systems where real-time control operations are required. It may seem that changing the operating system to a RTOS would convert the software to real-time control, but this is not the case in industrial control applications.

An RTOS only works in a limited number of control systems, because the interface betwen devices and control operations is specific to the given task. This organisation is not always compatible with a RTOS. Usually, RTOS kernels are do not offer portability because they may provide device drivers but only a fixed organization. Some parameterized RTOS kernels attempt to enhance portability by dividing the kernel into separate scheduling and communication layers. The communication layer abstracts the architecture-specific I/O facilities. The drivers for standard protocols are written once for each system and stored in a library. The scheduler is pre-emptive, priority-driven, and task-based. Whilst this is adequate for data-dominated applications with regular behaviors, this setup is incapable of handling the complex timing constraints often found in control-dominated systems. Instead of tuning code and delays, designers tune the priority assignments and design mostly by trial-and-error. To overcome the limitations of process-based scheduling, schedulers that focus on observable interrupts have been proposed. Real-time control can be enhanced by performing dependency analysis and applying code motion when the system is over-constrained by code-based constraints. Frame scheduling handles more general types of constraints by verifying that constraints are met in the presence of interrupts.

Software architecture is shown in Figure 1.8 for an embedded real-time control. It has one more layer, the Real-Time Shell, added into the embedded software architecture when compared with Figure 1.4. This is because the user interface layer, process handle layer, component control layer, and device driver/interface layer shown in Figure 1.4 are integrated into the user code layer of Figure 1.8.

Real-time systems should perform tasks in response to input interrupts. An interrupt can be a user request, an external, or an internal event. Real-time systems detect an interrupt by an interrupt mechanism rather than by polling. In embedded systems at run-time, a given interrupt may trigger different reactions, depending on the internal state of the system. We use the term "mode" to explain a given software architecture. A mode is a function that maps a physical interrupt with associated values onto a logical event; the system assigns each logical event a corresponding handler as defined by the user. The defined mode allows a given physical interrupt to take on different logical meanings by triggering an entirely different response or even to be ignored entirely.

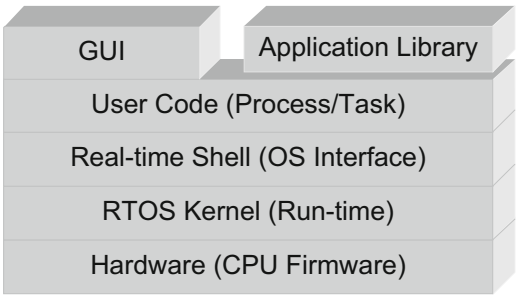


FIGURE 1.8

A software architecture for embedded real-time systems.

Each handler may also instruct the system to change mode. Specifically, if the mode waits on a timer interrupt, a self-transition effectively reinitializes the delay, while in the latter case the delay continues after handling an interrupt without a transition. Sequencing is a simple replacement of the current mapping with that of the target mode. This mapping can combine with additional (trigger, response) entries.

It is a way of composing modes to form hierarchical modes, because it combines the mapping functions of the submodes. Each submode can make its own transitions independently of the other submodes. Different submodes may also transition to a joining mode that does not become effective until all of its predecessors have completed their transitions to it. A join is the inverse of a fork. Finally, a disable kills other forked branches as well.

The real-time shell layer given in Figure 1.8 exists between user code and the RTOS Kernel. This additional layer separates the responsibilities between the upper and lower layers. The real-time shell is responsible for driving devices and I/O interfaces, handling interrupts, managing modes and coordinating timing. Thus this layer should contain the revelant program modules.

(a) Mode manager

The mode manager is divided into two distinct functions: mapping physical interrupts to interrupt handlers for dispatch, and for changing modes. An interrupt is reported by either the upper or the lower layer. Once the mode manager detects an interrupt, it first checks whether this interrupt is worth handling. If it is, it changes the current mode into that specific for handling this interrupt, and then maps this interrupt to the corresponding interrupt handler. The interrupt handler then triggers a task that handles this interrupt

Output requests are also received by the interrupt handler for scheduling. It informs the real-time task engine to disable the interrupts for the exiting mode, and enables the interrupts for the new mode. After the handler completes execution, if there is a mode transition to perform, the mode manager reconfigures the interrupt handler to reflect the current sensitivity list. To construct the new mapping, the mode manager applies one of the transition operators, which are sequencing, fork, join, or disable.

(b) Interrupt handler

Interrupt handling is a key function in real-time software, and comprises interrupts and their handlers. Only those physical interrupts which of high enough priority can be centered into system interrupt table. The software assigns each interrupt to a handler in the interrupt table. An interrupt handler is just a routine containing a sequence of operations. Each of these may request input and output while running.

The routine for handling a specific interrupt is known as the interrupt service routine for the specific interrupt. The RTOS layer often stores a list of the pairs of interrupts and their handlers known as the interrupt table. All interrupt handlers run constant within the background process. Most RTOS kernels issue tasks which can be triggered and terminated at run-time by the context switch paradigm. In object-oriented programming languages (such as C++ and Java), tasks can be run as one or more threads. Thus, an interrupt can be handled either as a thread or as a sub-process within a task or process.

(c) Real-time task engine

Interrupts from different device drivers are assigned different priorities.

There are two object queues held in the RTOS shell program. The first is the interrupt calendar queue and the second is the task queue. The interrupt calendar queue is ordered in terms of interrupt

priorities and the first-come first-served principle. The task queue is a key object in the RTOS shell. It is held in the form of a list or a table structure, and holds the following types of tasks: main task, background task, current task, and other tasks. In the task queue, tasks are also ordered in terms of their priorities and the first-come first-served principle.

While running a task at run-time, interrupts arrive. The interrupt handler first checks whether it is a valid interrupt. If it is, the interrupt handler first puts it into the interrupt calendar queue. Then, the interrupt handler picks up the interrupt at the head of the interrupt calendar queue and refers to the interrupt table for corresponding service routine. At this moment, the dispatcher uses the RTOS kernel context switch to stop the current task and store all its data in memory (ROM). Control then transfers to the scheduler, which selects a task from the task queue to run the interrupt service routine.When a task finishes a complete run, the dispatcher takes over, and then transfers to the scheduler to get the next task from the task queue.

There are several ways to approach software for real-time control systems. The hardware layer codes include the machine-instruction set hardcoded inside the CPU firmware. A range of RTOS kernels or nuclei, often written in both assembler and C languages are available. Choice is usually based on the microprocessors used. User code will then need to be developed in a real-time control system. Object-oriented programming languages, such as C++ and Java as well Ada, are the programming languages most often used program.

Code development for a given real-time control system will be a complex job. Usually the process involves iterating the following steps: program design including classes, sequences and user case designations; coding programs in accordance with design; performing simulation tests; design modification; coding the program again in terms of the modified design; and repeat until a satisfactory result is obtained.

(3) Real-time verification and tests

In development make mistakes are often made in programming real-time systems. These programming errors are seldom, visible in the debugging phase, hence can appear when the application is running and it is not possible to correct the mistakes. Thorough necessary debugging before the software is delivered.

(a) Program bugs

A program bug is a flaw, fault, or error that prevents the program from executing as intended. Most are the results of mistakes and errors made by programmers in either source code or program design, and a few are caused by compilers producing incorrect code.

Control switching statements (if-then-else or case-switch statements) occur often in program code. These statements allow different paths of execution of varying length, so code will have different execution times. This becomes more significant when the path is very long. Variable functions, state machines and lookup tables should be used instead of these kinds of statements.

Some programs contain a single big loop such as a For or a Do While loop. This means that the complete unit always executes at the same rate. In order to assign different and proper rates, concurrent techniques for a pre-emptive RTOS should be used.

Some code as message passing as the primary intertask communication mechanism. In real-time programming, message passing reduces real-time scheduling (the maximum value of the system's CPU utilization for a set of tasks up to which all tasks will be guaranteed to meet their deadline) and produces

significant overheads, leading to many aperiodic instead of periodic tasks. Moreover, deadlocks can appear in closed-loop systems, which could cause execution to fail. Shared memory and proper synchronization mechanisms are used to prevent deadlocks, and priority inversion should be used.

(b) Incorrect timing

Many RTOS kernels provide a program delay, running or waiting mechanism for hardware operations, such as I/O data transfer and CPU cleaning of memory partitions. They are implemented as empty or dummy loops, which can be lead to non-optimal delays. RTOS timing mechanisms should be used here for implementing exact time delays.

Program design should be done with execution-time measurement. It is very common to assume that the program is short enough, and the available time is sufficient, but measuring of execution time should be part of the standard testing in order to avoid surprises. The system should be designed such that the execution timing of code is always measurable.

(c) Misuse interrupts

In a real-time system, interrupts cannot be scheduled by the scheduler because interrupts are mandatory for getting services. Because they can occur randomly, interrupts seriously affect real-time multi-tasking predictability. Programs based on interrupts are very difficult to debug and to analyze. Moreover, interrupts operate in the task context handled by the RTOS kernel; a task corresponding to the response to a given event could lack a real-time timing requirement.

There is a very common misunderstanding which says that interrupts save CPU time and they guarantee the start of execution of a task. This can be true in small systems, but it is not the case for complex real-time systems, where non-pre-emptive periodic tasks can provide similar latency with better predictability and CPU utilization. Therefore, interrupt service routines should be programmed in such a form that their only function is to signal an aperiodic server.

(d) Poor analyses

Memory use is often ignored during program design. The amount of memory available in most real-time systems is limited. Frequently, programmers have no idea how much memory is used by a certain program or data structure. A memory analysis is quite simple in most real-time system development environments, but without it a program can easily crash at run-time.

Sometimes, statements are used to specify register addresses, limits for arrays, and configuration constants. Although this practice is common, it is undesirble as it prevents on-the-fly software patches for emergency situations, and it increases the difficulty of reusing the software in other applications. In fact, changes in configuration require that the entire application has to be recompiled.

1.3 DISTRIBUTED CONTROL SYSTEMS

1.3.1 Principles and functions

The distributed control system (DCS) is a concept which is difficult to define. To fully clarify what a distributed control system is, it is helpful to understand the evolution of control system implementation and hardware elements, and how information flow and decision-making developed.

In industry, a control system is a device or set of devices to manage, command, direct or regulate the behavior of other devices or systems. The first type of industrial control system to evolve was a centralized control system in which all information processing is carried out by one particular controller. Figure 1.9 is a graphical description of a centralized control system for automatically shutting-off all lights inside a building. In this system, control panels are used to provide a central platform for switching large numbers of loads. They contain relays with low-voltage inputs from control devices and line-voltage outputs to the load. The system is centralized, with all local switches and switch-legs (sub-circuits) connected to the control panel via line-voltage wiring, and accessory inputs such as photoelectric sensors connected to the panel via low-voltage control wiring. The control panel polls connected control devices for input, which is then filtered through its logic circuit to determine the output (ON or OFF).

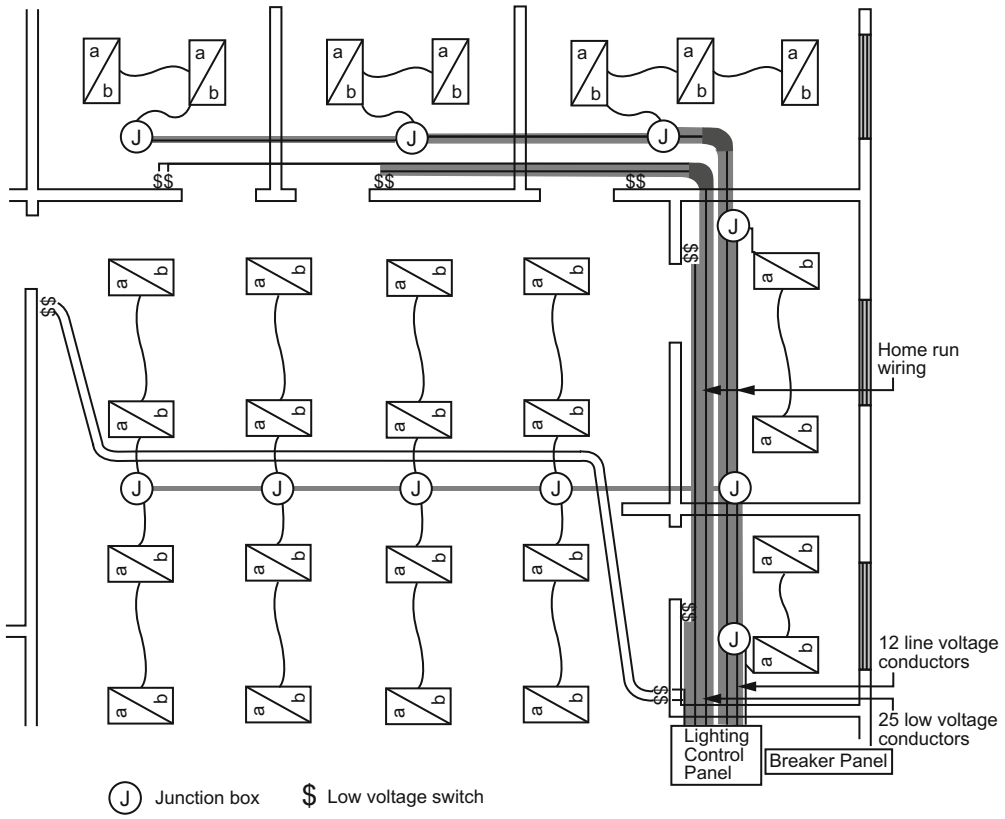


FIGURE 1.9

A centralized control system for automatic shut-off of all lights of a big building, in which the wiring requires switches and line-voltage switch-legs to be individually connected to the central control panel.

(Courtesy of the Watt Stopper and Legrand.)

Figure 1.10 is another example; this time structured as a distributed control system. In this approach, panel-based control functionality is distributed across the facility. An example is when a control panel is installed on each floor of a multi-story building, and these are networked for control from a master panel, or for convenient access and individual stand-alone control by each floor's occupants. A completely distributed system would have no control panel in the electrical room. The distributed panels are typically networked via low-voltage cabling to share information, and implement local and global commands based on a shared protocol. For centralized scheduling, these distributed panels can connect back to a system time-clock.

When dealing with real applications of control systems, the location that data are acquired from, and the location to which decisions we conveyed, are both fixed. In contrast, where data processing or decision-making occurs is almost completely arbitrary. In distributed control, processing should be performed where it is most feasible for the goals of the overall system. With this understanding, the

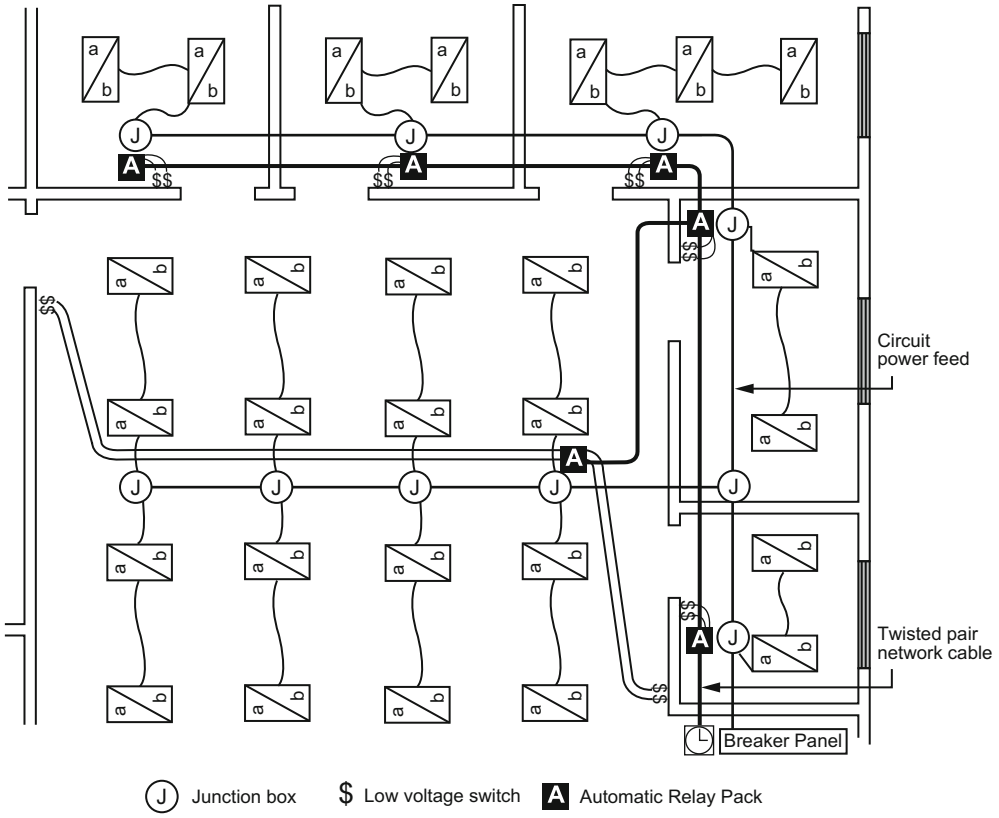


FIGURE 1.10

A distributed control system for automatic shut-off of all lights of a big building, which requires only a twisted-pair communication cable to be run between devices and back to a system time-clock.

(Courtesy of the Watt Stopper and Legrand.)

control system can be considered as an abstraction into which data enter, are combined and manipulated for some purpose, and are then released back into the environment, which is subsequently altered. The process is then endlessly repeated. The rate at which this occurs is a critical parameter in the efficient functioning of the system.

There are very few systems which can be classified as entirely centralized. For example, many field elements perform some processing, although analog to linearize the data. While a linearized element at field level is not typically considered to be a distributed system element, it does in fact relieve some of the processing burden from a control rule processor. In order to share information between processes, most effectively however, the data must be available in a format which is consistent and unambiguous. Standardized data also require the least amount of manipulation by the receiving process in order to be used. Figure 1.11(a) shows these rationales of a centralized control system. Supervisory control and data acquisition (SCADA) systems are typical examples of a centralized control system.

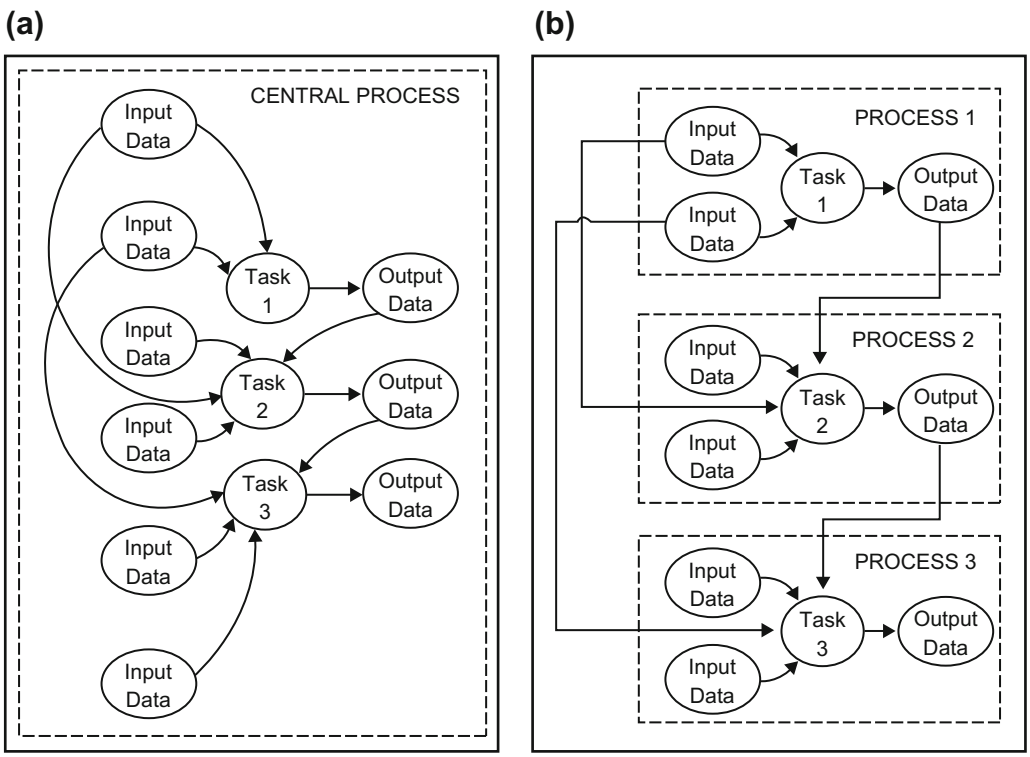


FIGURE 1.11

A graphic explanation of the working principles of industrial control systems. (a) A centralized control process in which each task shares processing time, thus data flow is controlled within the process. (b) A distributed control process in which each task is run independently and simultaneously, thus data flow must be carefully coordinated between processes.

In the context of some architectures of distributed control systems, the single most important criterion for effective control is data availability. This assumes that each distributed process requires data from a neighboring process to complete its task. To share information between processes most effectively, the data must be available in a format which is consistent and unambiguous. Distributed control systems are those in which processing is performed at the location most beneficial to the overall system goals; information flows throughout the system and is available wherever and whenever it is needed. Information is available in a standard format which is unambiguous to the receiving process. Figure 1.11(b) shows these rationales of a distributed control system.

A commonly accepted definition of a distributed control system is a system which uses decentralized elements or subsystems to control distributed processes, or complete manufacturing systems. They do not require user intervention for routine operation, but may permit operator interaction via a supervisory control and data acquisition (SCADA) interface.

Distributed processing and distributed control are becoming unified in control architectures. In distributed processing, functions such as signal processing and data acquisition are distributed, but the control may be centralized, which is hybrid distributed control. In fully distributed control, both processing and the control function are distributed. Thus, we can say that distributed control also implies distributed processing.

Distributed control systems can be classified as open or closed. Open distributed control systems consist of many devices within an open network, which are tuned more to open-source development. On other hand, closed distributed control systems have the devices interconnected within a closed network, which are tuned to traditional, single baseline development. Although distributed control systems are closely linked to device networks, the two are not equivalent. For example, a sensor network with one controller only is not said to be a distributed control system.

In more complex pilot plants and full-scale plants, the control loops number in the hundreds. For such large processes, a computerized distributed control system is more appropriate. Conceptually, computerized DCS is similar to a simple PC network. However, there are some differences. In DCS architecture, the microcomputers attached to the process are known as front-end computers, and are usually less sophisticated pieces of equipment employed for low-level functions. Basically, various parts of the plant processes and several parts of the computerized DCS network elements are connected to each others by means of a data highway (Fieldbus). Typically such equipment would acquire process data from the measuring devices, and convert them to standard engineering units. The results at this level are passed upward to the larger computers that are responsible for more complex operations. These upper-level computers can be programmed to perform more advanced calculations and decision-making.

A schematic of computerized DCS is shown in Figure 1.12, in which only one data highway is shown. The data highway is the backbone of the computerized DCS system. It provides information to multiple displays on various operator control panels, sends new data and retrieves historical data from archival storage, and serves as a data link between the main control computer and other parts of the network. At the top of the hierarchy, a supervisory (host) computer is set. This is responsible for performing higher-level functions. These could include optimization of the process operation over varying time-scales, carrying out special control procedures such as plant start-up or product grade transition, and providing feedback on economic performance.

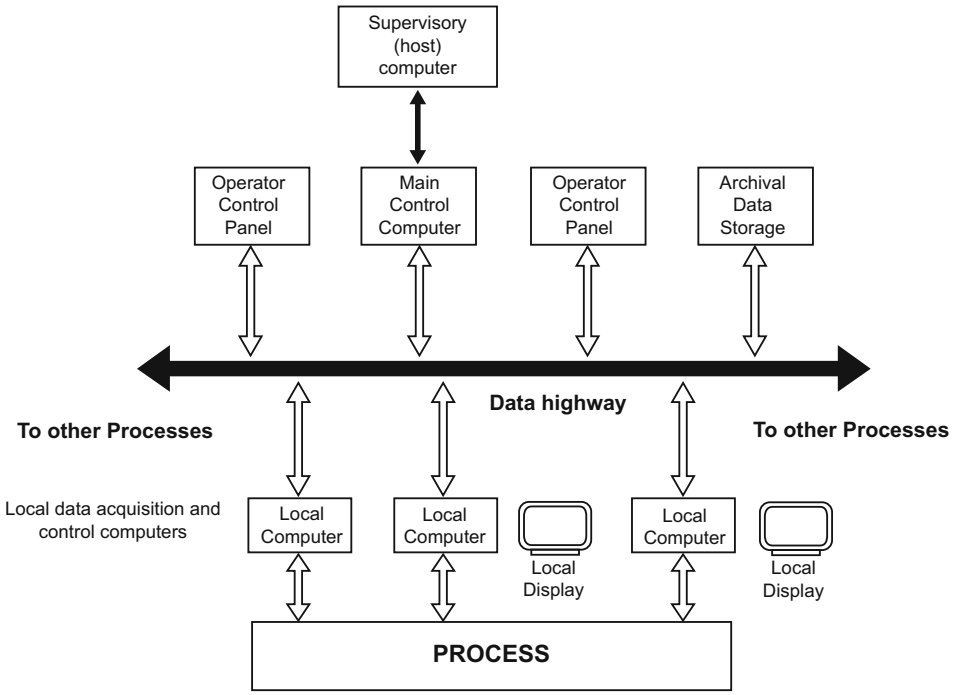


FIGURE 1.12

Schematic of a computerized distributed control system.

1.3.2 Architectures and elements

Some distributed control systems are used in electrical power grids or electrical generation facilities. Others are used in environmental control systems, wastewater treatment plants, and sensor networks. Distributed control systems for petroleum refineries and petrochemical plants are also common.

In industrial applications, different structures of distributed control systems (DCS) exist. They include fully distributed control, hybrid distributed control, open distributed control, closed distributed control, computerized distributed control, wireless distributed control, and so on. Each type has a different DCS architecture. Figure 1.13 is a graphic configuration of all distributed control system architectures. In the diagram, the upper box shows fully distributed and closed architectures (where centralized architecture means centralized distributed architecture, which is different from the centralized control system architecture mentioned earlier); the middle part depicts wireless connections in distributed control systems; the lower box shows that there are control networks in distributed control systems. These control networks include local area networks (LANs), SCADA networks, PLC networks, etc. The whole of Figure 1.13 illustrates an open and hybrid distributed control system architecture.

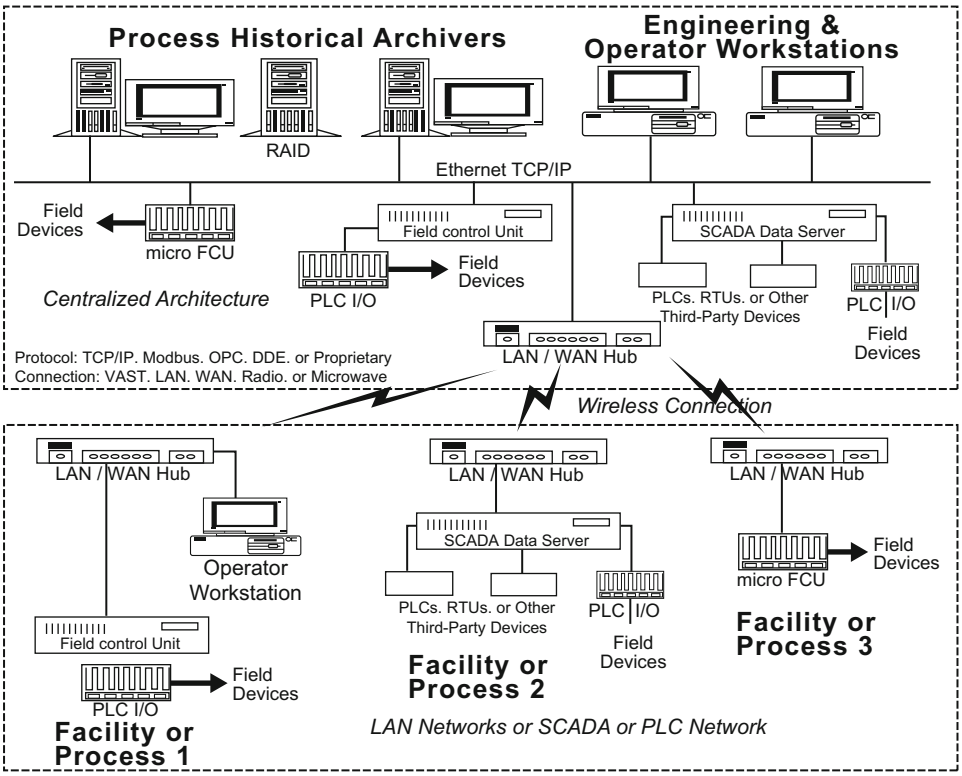


FIGURE 1.13

Distributed control system architecture.

(Courtesy of Control Systems International, Inc.)

(1) Hardware components

(a) Operator interfaces

The operator interface features high-resolution graphics and graphical user interface (GUI) windows. Its responsibilities are: to allow operators to monitor detailed activities of many types of device and send commands using command windows and group displays; to display project screens created during project development and makes them dynamic based on real-time data; to log and display alarms and other event messages, and let operators acknowledge alarms; to show trends in real-time and historical data; to allow authorized operators to monitor and override device logics; to display the status of communication between the devices and components, which helps the operator diagnose the cause of communication interrupts.

(b) I/O subsystem

It is necessary for distributed control systems to support standard, off-the-shelf I/O subsystems, so that the same logic can be used to manipulate different I/O subsystems without the need to change

programming or operational parameters of the configured system. This allows a distributed control system to replace another without requiring replacement of the I/O subsystem. In addition, when the I/O subsystem is replaced, no change in logic is required since distributed control system logic can be hardware-independent.

(c) Connection buses

In distributed control systems, Ethernet is a local area network (LAN) protocol that uses a bus or star typology and supports data transfer rates of 10 Mbps. To handle simultaneous demands, Ethernet uses carrier sense multiple access/collision detection (CSMA/CD) to monitor network traffic.

Fieldbus is a bi-directional communication protocol used for communications among field instrumentation and control systems. The process Fieldbus (PROFIBUS®) is a popular, open communication standard used in factory automation, process automation, motion control, and safety applications.

Network protocols for distributed control systems (DCS) also include controller area network bus (CANbus), control network (ControlNet), DeviceNet, INTERBUS, and PROFIBUS, and other protocols.

(d) Field control units

Field elements are controlled by PLC, PC or remote terminal units running under any operating system. The controller software executes sequential and regulatory control schemes that are downloaded from the engineering workstation. It also provides real-time scanning and control of remote I/O.

The SCADA Data Server is another component to communicate with devices that cannot be directly scanned by the PLC or remote terminal units. The SCADA data server is a generalized, flexible, and user-configurable application designed to interface to other control systems, proprietary protocol drivers, or specialized software.

(e) Wireless subsystem

If replacing Ethernet and some connection buses with wireless Ethernet and wireless connections in a distributed control system, it will thereby contain a wireless subsystem. In most applications, wireless subsystems are used for remote communication between parts of the distributed control system which are physically distant.

(f) Component redundancy

Most advanced distributed control systems support selective redundancy at all levels, including redundant field controllers, field buses, and redundant I/O components.

(2) Software modules

(a) History module

This module stores the configurations of the distributed control system, as well as the configurations of all the points in the controlled system. It also stores the graphic files that are shown on the console and in most systems. These files can be used to store some operating data of the control systems and of the controlled systems. All the data stored and managed by the history module can be kept on hard disks in the PC used as the operator interface.

(b) Control modules

These modules carry out control functions on the target systems. They are the brain of a distributed control system. Custom function blocks are especially found in these modules, and are responsible for operating control algorithms such as PID, fuzzy-logic, ratio control, simple arithmetic and dynamic compensation, and more advanced control features.

(c) I/O modules

These modules manage the input and output of a distributed control system. The input and output can be digital or analog.On and off, start and stop signals are examples of digital I/O modules. Most process measurements and controller outputs are I/O modules analog. These points are hardwired with the field elements.

(3) Network models

(a) The Ethernet LAN model

Ethernet is the most common physical layer in distributed control systems, but it is also used as the transport mechanism in a distributed control system. The Ethernet network protocols are referred to as TCP/IP. This set of protocols accommodates different kinds of data for different applications.

Ethernet provides communication at a faster data rate on an office LAN. A remote hard drive is just as accessible as a local one because the network is able to deliver the requested data quickly. Fast Ethernet 100BASE-Y and Gigabit Ethernet, are the two kinds of Ethernet most used in distributed control systems.

Fast Ethernet 100BASE-T is 10BASE-T with the original Ethernet Media Access Controller (MAC) at 10 times the speed. It allows three physical-layer implementations, all part of IEEE 802.3u: 100BASE-TX, which has two pairs of Category 5 UTP or Type 1 STP cabling and is most popular for horizontal connections; 100BASE-FX, which has two strands of multimode fibre and is most popular for vertical or backbone connections; 100BASE-T4, which has four pairs of Category 3 or better cabling and is not common.

Gigabit or 1000-Mb Ethernet is the 1998 IEEE 802.3z standard and includes the Gigabit Ethernet MAC and three physical layers. Gigabit uses 8B/10B encoding and encompasses four physical standards: 1000BASE-SX Fibre (horizontal fibre), 1000BASE-LX Fibre (vertical or campus backbone), 1000BASE-CX Copper, and 1000BASE-T.

(b) The OSI network model

The standard model for networking protocols and distributed applications is the International Standard Organization's Open System Interconnect (ISO/OSI) model. It defines seven network layers.

Layer 1 is Physical. The physical layer defines the cable or physical medium itself. All media are functionally equivalent. The main difference is in convenience and cost of installation and maintenance. Converters (digital to analog and vice versa) from one media to another operate at this level.

Layer 2 is Data Link. The data link layer defines the format of data on the network: a network data frame, or packet, includes checksum, source and destination address, and data. The largest packet that can be sent through a data link layer defines the Maximum Transmission Unit (MTU). The data link layer handles the physical and logical connections to the packet's destination, using a network interface. A host connected to an Ethernet would have an Ethernet interface to handle connections to the outside world, and a loopback interface to send packets to it.

Layer 3 is Network. NFS uses Internetwork Protocol (IP) as its network layer interface. IP is responsible for routing, and directing a datagram from one network to another. The network layer may have to break large datagrams, larger than the MTU, into smaller packets, and the host receiving the packet will have to reassemble the fragmented datagram. The Internetwork Protocol identifies each host by a 32-bit IP address. IP addresses are written as four, dot-separated, decimal numbers between 0 and 255. The leading 1-3 bytes of the IP address identify the network, and the remaining bytes identify the host on that network. For large sites, the first two bytes represents the network portion of the IP address, and the third and fourth bytes identify the subnet and host respectively. Even though IP packets are addressed using IP addresses, hardware addresses must be used to actually transport data from one host to another. The Address Resolution Protocol (ARP) is used to map between IP addresses and hardware addresses.

Layer 4 is Transport. The transport layer subdivides the user-buffer into network-buffer sized datagrams and enforces desired transmission control. Two transport protocols, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), sit at the transport layer. Reliability and speed are the primary difference between the two protocols. TCP establishes connections between two hosts on the network through sockets, which are determined by the IP address and port number. TCP keeps track of packet delivery order and of which packets that must be resent. Maintaining this information for each connection makes TCP a state protocol. UDP on the other hand provides a low overhead transmission service, but with less error checking. NFS is built on top of UDP because of its speed and statelessness. Statelessness simplifies crash recovery.

Layer 5 is Session. The session protocol defines the format of data sent over the connections. NFS uses the Remote Procedure Call (RPC) for its session protocol. RPC may be built on either TCP or UDP. Login sessions use TCP, whereas NFS and broadcast use UDP.

Layer 6 is Presentation. External Data Representation (XDR) sits at the presentation level. It converts a local representation of data to its canonical form, and vice versa. The canonical form uses a standard byte ordering and structure packing convention, independent of the host.

Layer 7 is Application. This layer provides the network with services to the end-users. Mail, FTP, telnet, DNS and NFS are examples of network applications.

(c) The TCP/IP network model

It is generally agreed that the TCP/IP model should have fewer levels than the seven of the OSI model. Most descriptions present from three to five layers. The structure of TCP/IP is built as information is passed down from applications to the physical network layer. When data are sent, each layer treats all of the information it receives from the layer above as data, and puts control information at the front of that data. This control information is called a header, and the addition of a header is called encapsulation. When data are received, the opposite procedure takes place as each layer removes its header before passing the data to the layer above. This textbook gives a TCP/IP model of four layers.

The top layer of the TCP/IP model is the application layer. In TCP/IP the application layer includes the functions of OSI presentation layer and session layer. In this book, we follow the convention that an application is any process that occurs above the transport layer, including all processes that involve user interaction. The application determines the presentation of the data, and controls the session. In TCP/IP the terms, socket and port are used to describe the path over which applications communicate. There are numerous application-level protocols in TCP/IP, including Simple Mail Transfer Protocol (SMTP) and Post Office Protocol (POP), which are used for e-mail, Hyper Text Transfer Protocol

(HTTP) used for the World-Wide Web, and File Transfer Protocol (FTP). Most application-level protocols are associated with one or more port number.

The layer below the application layer is the transport layer. In the TCP/IP model, there are two transport layer protocols. The Transmission Control Protocol (TCP) guarantees that information is received as it was sent. The User Datagram Protocol (UDP) performs no end-to-end reliability checks.

The layer below the transport layer is the internet layer. In the OSI Reference Model the network layer isolates the upper layer protocols from the details of the underlying network, and manages the connections across the network. The Internet Protocol (IP) is normally described as the TCP/IP network layer. Because of the inter-networking emphasis of TCP/IP this is commonly referred to as the internet layer. All upper and lower layer communications travel through IP as they are passed through the TCP/IP protocol stack.

The lowest layer is the network access layer. In TCP/IP the data link layer and physical layer are normally grouped together. TCP/IP makes use of existing data link and physical layer standards rather than defining its own. The data link layer describes how IP utilizes existing data link protocols such as Ethernet, Token Ring, FDDI, HSSI, and ATM. The characteristics of the hardware that carries the communication signal are typically defined by the physical layer. This describes attributes such as pin configurations, voltage levels, and cable requirements. Examples of physical layer standards are RS-232C, V.35, and IEEE 802.3.

1.3.3 Implementation techniques

Distributed control systems (DCS) differ in terms of complexity and applications. Smaller implementations may consist of a single programmable logic controller (PLC) connected to a computer in a remote office. Larger, more complex DCS installations are also PLC-based, but use special enclosures for subsystems that provide both I/O and communication functionalities. Fully distributed systems enable remote nodes to operate independently of the central control. These nodes can store all of the process data necessary to maintain operations in the event of a communications failure with a central facility.

Distributed control systems consist of a remote control panel, communications medium, and central control panel. They use process-control software and an input/output (I/O) database. Some suppliers refer to their remote control panels as remote transmission units (RTU) or digital communication units (DCU). Regardless of their name, remote control panels contain terminal blocks, I/O modules, a processor, and a communications interface. The communications medium in a distributed control system is a wired or wireless link which connects the remote control panel to a central control panel, SCADA, or human machine interface (HMI). Specialized process-control software is used to read an I/O database with defined inputs and outputs.

A major problem with distributed systems is the lack of a proper use of standards during system development. A common hardware and software interface interconnecting the elements of the system should be specified. The definition of a common interface standard benefits both the decomposition of the system control problem and the integration of its components. Distributed systems do not require common interfaces but the benefits of such an implementation should be evident. The use of Open Systems Interconnection (OSI) standards is most preferable, because it increases participation in the development of the interface beyond an individual engineering project. In other words, any unique or proprietary solution to a problem has limited support because of resource limitations. OSI increases the

availability of support hardware and software and leads to a better understood, robust, and reliable system.

What are some of the technical challenges in distributed control systems? The following is a partial list.

(1) Partitioning, synchronization, and load balancing

For certain applications, optimal partitioning of the architecture into distributed nodes may be difficult to determine. The need for synchronization between distributed control programs influences the degree of distribution. Once a set of distributed nodes are identified, the control program needs to be partitioned. Program partitioning should consider static and dynamic balancing of processing load in the distributed nodes.

(2) Monitoring and diagnostics

In a centralized control system, there is a central repository of sensor and actuator data. In such systems, it is easy to construct the state of the process. Process monitoring and failure diagnosis is often performed by observing the temporal evolution of the process state. However, monitoring and diagnosing process failures in distributed control systems will require new methods for aggregating the system state from distributed data.

(3) Fault-tolerance and automatic configuration

We have to architect the system for distributed fault-tolerant operation. If a distributed node fails, the system should stay operational albeit with reduced performance. Such systems will need real-time distributed operating systems and communications technology for adding and removing control nodes without shutting the system down.

Several promising new technologies can provide innovative solutions to the above challenges. The application of graphical programming, human machine interface and simulation technology can simplify programming, debugging and configuration of distributed control systems. For discrete-event systems, methods from Discrete-Event Dynamic Systems (DEDS) theory can be applied to partition and verify the control code and construct observers for diagnostics. Autonomous agents' technology can be applied to automatic reconfiguration and negotiation between nodes for load sharing. Distributed real-time operating systems will provide the necessary services for implementing the distributed control architecture.

(4) Comprehensive redundancy solutions

To achieve a high level of reliability, many DCS have been designed to have built-in redundancy solutions for all control levels from a single sensor to the enterprise-scale servers. Redundancy features for most components of a distributed control system are provided automatically and no additional programming is required. A comprehensive solution for redundancy in distributed control systems may include the following features.

(a) Redundancy solution for sensors

The reliability assurance system, which can be a subsystem of some controllers, allows control of the quality of signals acquired from sensors and provides redundancy. In the case of a break in communication with the sensors equipped with digital interfaces, hardware invalidation will be

stated for all signals received from them. Flags for hardware and software invalidation are transmitted in the trace mode channels along with the value measured, as one of the channel attributes, and they can be used in algorithms which can be flexibly tuned by users. There are no limitations whatever for redundancy of sensors or groups thereof (for example, I/O cards). Dual and triple redundancy systems may be built easily. Trace mode may therefore be used to build control systems that monitor the sensor signal quality in real time, and provide redundancy features that increase the overall system reliability.

(b) Redundancy solution for controllers

This type of redundancy is used, as a rule, to ensure reliability of control systems in hazardous processes. Controller redundancy algorithms can be flexibly adjusted by the user and corrected in compliance with the requirements of a particular control system.

By default, the following hot redundancy technologies are implemented in trace mode: a channel database is automatically built for the standby controller; controller redundancy is performed by switching over of data flows to the standby controller in real time; synchronization of real-time data between the main and standby controller.

Dual or triple redundancy provision for controllers does not exclude duplication of sensors. Sensor redundancy may be provided for every signal. The user can determine whether each of the dual redundant controllers will get data of the given technological parameter from its sensor, or whether one sensor is to be used as a source of information for both the controllers.

The reliability assurance system also includes Watch Dog timer support, which helps to reboot controllers and industrial PCs automatically in the case of a system halt.

If another PLC programming system is used instead of micro trace mode, control system reliability is ensured as follows; the hardware validation flag would be generated at the human machine interface of the PC, indicating availability of communication with the PLC. If the PLC or server supports signal quality control, the communication quality flag may be input in a separate trace mode channel. This channel value will be considered by the trace mode for hardware validation/invalidation flag generation. Such a technique helps in developing the template algorithms for signal redundancy (or for groups of signals) in the PLC or in the fail-proof trace mode servers.

(c) Redundancy solution for connection buses

If a system is connected through two bus interfaces (gateways) to a bus (line) of either one or two masters, this redundancy concept does not allow for a redundant bus structure. The switch-over of the master module is not coupled to the slave module. The slave must be applied separately to both masters, so that double the engineering effort may be required with a partial enhancement of the availability.

On the other hand, a twin bus structure provides system redundancy. In this concept two masters communicate through separate bus structures with a slave, which also features two gateways. For a long time no standard existed in the PROFIBUS regarding the implementation of system redundancy. The implementation of various manufacturer-specific redundancy concepts is partly not possible or requires an inordinate amount of engineering effort in the control system. The question is immediately posed whether applications of this type can be maintained in the control system. The user usually prefers to do without a redundant PROFIBUS because control system software cannot be maintained.

(d) Redundancy solution for the I/O interface

Network path redundancy is accomplished by connecting a ring topology and providing alternative cable path among the modules, should a physical break in the ring occur. I/O point redundancy can be achieved by adding modules to the network, with an operating algorithm to ensure correct operation. For example, three input modules may measure the same temperature, and the input temperature is measured as their average, provided all three are within acceptable limits; when any one is outside a predetermined difference from the other two, the two similar readings are believed and a diagnostic is flagged in the system.

Problems

1. Find several electronic devices or systems from your observations in home, supermarkets, railway stations and trains, etc. that you think are possibly embedded control systems.
2. Analyze the automatic gear shift function of automobiles for the elements and architectures of its hardware system and mechanical system, then drew their diagrams with reference to Figure 1.1 and Figure 1.3.
3. If designing the software for the automatic gear shift function of automobiles, do you prefer the layers model or the classes model? Explain the reasons for your preference.
4. Do you believe that real time control systems are those computerized and digital control systems in which real time algorithms are implemented? Please explain your answer.
5. Give all the items included by "real time algorithm"; and explain how can these items support real time controls.
6. Think about the real time control systems in car steering and braking; should they be hard or soft real time control? Think about the real time control system of a cash machine; should it be a hard or soft real time control? Please explain your answers.
7. Analyses of the working mechanisms of two control systems for automatic shut off lights in a building are given in Figure 1.9 and 1.10, respectively; explain why the system in Figure 1.9 is called centralized control, and the system in Figure 1.10 is called distributed control. Then, combine Figure 1.9 with Figure 1.11 (a) and Figure 1.10 with Figure 1.11 (b) to specify the contents of input data, output data, tasks and process.
8. Indentify the computerized distributed control system given in Figure 1.12 as a fully or a hybrid distributed control system.

Further Reading

Wikipedia (http://en.wikipedia.org). 2008A. Embedded systems. http://en.wikipedia.org/wiki/Embedded system#Examples of embedded systems. Accessed: January 2008.

World Association of Technology Teachers (http://www.technologystudent.com). 2006. Computer control index. http://www.technologystudent.com/comps/compdex.htm. Accessed: January 2008.

Jan F. Broenink, Gerald H. Hilderink. A Structured approach to embedded control systems implementation. Proceeding of 2001 IEEE International Conference on Control Applications, September 5 7, 2001; Mexico City. pp. 761 766.

A. Gambier. Real time Control Systems: A Tutorial. IEEE Control Conference, July 20 23, 2004; Tokyo. 5th Asian Volume 2; pp. 1024 1031.

Spyros G. Tzafestas, J. K. Pal (Eds.). Real Time Microcomputer Control of Industrial Processes. Dordrecht: Kluwer Academic. 1990.

J. W. S. Liu. Real time Systems. New York: Prentice Hall. 2000.

P. Chou, G. Borrielo. Software architecture synthesis for retargetable real time embedded systems. Proceeding of the Fifth International Workshop on Hardware/Software Code Designs. March 24 26, 1997. Braunschweig, Germany. pp. 101 105.

Dennis E. Culley, Randy Thomas, Joseph Saus. Concepts for distributed engine control. NASA/TM 2007 214994. AIAA 2007 5709. Glenn Research Center, OH, USA.

Craig Dilouie. Distributed control systems promise greater value. Lighting Control Association: http://www.aboutlightingcontrols.org/education/papers/distributed control.shtml. Accessed: February 2006.

Xiangheng Liu, A. Goldsmith. Wireless network design for distributed control. 43rd IEEE Conference on Decision and Control. December 14 17,2004. Volume 3, pp. 2823 2829.

Control Systems International, Inc. UCOS Functional Overview: Version 5.0. http://www.ucos.com/UCOSFunctionalOverview.pdf. Accessed: February 2008.

G. J. Beider, R. W. Wall. Low cost distributed control system for autonomous vehicles. IEEE Proceeding of OCEAN 2006 Asia Pacific. Singapore. May 16 19, 2007.

CHAPTER

Industrial control engineering 2

Industrial control engineering is the branch of industrial engineering that deals with the design, development, and implementation of integrated systems of humans, machines, devices, and information resources which together provide applications that behave in the desired manner. Industrial control engineering encompasses both control theory and control technology. Thus, it involves modeling the spatial and temporal natures of industrial objects; analyzing the dynamic behavior of industrial objects; using control theory to create control strategies; designing control devices, hardware and software; and implementing industrial control systems.

Industrial control engineering is a very complex subject, with its main areas being work methods analysis and improvement; work measurement and the establishment of standards; machine tool analysis and design; workplace design; plant layout and facility design; materials handling; cost reduction; production planning and scheduling; inventory control, maintenance, and replacement; statistical quality control; scheduling; assembly-line balancing, systems, and procedures; and overall productivity improvement. Computers and information systems have necessitated additional activities and functions, including numerically controlled machine installation and programming, manufacturing systems design, computer-aided design and computer-aided manufacturing, design of experiments, quality engineering, and statistical process control, computer simulation, operations research, and management science methods; computer applications, software development, and information technology; human-factors engineering; systems design and integration; and robotics and automation.

Modern industrial control engineering can be categorized into three areas: process control, motion control, and production automation. This chapter will focus on these three areas, and will explain their system models, control strategies, control devices, and control implementations.

2.1 INDUSTRIAL PROCESS CONTROLS

2.1.1 Definition and functions

Industrial processes in this context means those procedures in industries which involve chemical reactions, material changes, or mechanical steps which are carried out to effect the transport or manufacture of objects, usually on a very large scale.

There are many examples of industrial processes, including energy transmission, liquid flows, metal forges, chemical reactions, part cutting etc.

Industrial process control can be specified as the automatic monitoring and functioning of an industrial process, brought about by controllers or computers with programmed strategies and algorithms, associated with some devices and instruments to appropriately respond to feedback from the

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10002-6

process. Since the late twentieth century, the field of industrial process control has become increasingly important in chemical, pharmaceutical and petrochemical plants, oil refineries, energy transmission, and other related industries. This is because industrial process control can help industries to improve product quality, enhance production rates, stabilize plant and device operations, reduce working costs, minimize environmental pollution, etc. Advanced industrial process control in practise can be characterized as one or more of the following four types.

(1) Discrete process controls

Discrete industrial processes specify those industrial processes which handle distinct, separate products. In a discrete industrial process, both inputs and outputs must be discrete data flows and/or discrete data stores. Discrete processes can be found in many manufacturing, motion and packaging applications. The usual example is an automotive factory product line, where each car is a distinct artifact. Most discrete manufacturing involves the production of discrete pieces of product, such as metal stamping. Robotic assembly, such as that found in automotive production, can be characterized as discrete process control.

(2) Continuous process controls

Some industrial processes are characterized by smooth and uninterrupted variables in time, and are defined as continuous processes. Some important continuous processes are the production of fuels, chemicals and plastics, forging, cutting, heating, etc. Continuous process control is commonly applied to these continuous processes. In a continuous process control system, the variables associated with any process are monitored and subsequent control actions are implemented to maintain variables within predetermined process constraints. Figure 2.1 is a simple continuous process control system.

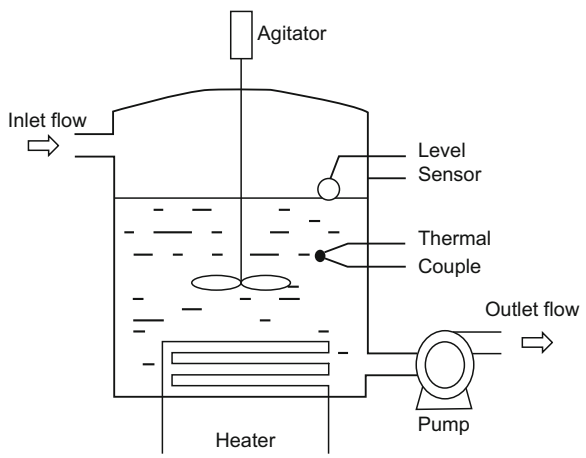


FIGURE 2.1

An illustration of continuous process control: this figure shows an agitated and heated tank reactor with a pump, a thermocouple, and a level sensor.

(3) Batch process controls

In addition to discrete and continuous processes, there exists another industrial process known as the batch process. Batch processes are characterized by a prescribed processing of materials for a finite duration. This contrasts with continuous processes, where reactants are continually fed in, and products taken out. Batch industrial processes can be of two kinds; the first is the batch production process that leads to the production of specific quantities of product, by subjecting quantities of input materials to an ordered set of processing activities over a finite period using one or more pieces of equipment; the second is the batch chemical process in which a measured quantity of reactants is added to a reaction vessel, the reaction is carried out, and the products are then removed. Batch process control plays an important role in the chemical industry where low-volume, high-value products are required. Examples include reactors, crystallizers, injection molding processes, and the manufacture of polymers. Temperature and pressure profiles are implemented with servo-controllers, and precise sequencing operations are produced with tools such as programmable logic controllers (PLC).

(4) Statistical process controls

Statistical process control (SPC) is a control method for monitoring an industrial process through the use of a control chart. Much of its power lies in its ability to monitor both the process center and its variation about that center. By collecting data from samples at various temporal and spatial points within the process, variations in the process that may affect the quality of the end product or service can be detected and corrected, thus reducing waste and the likelihood that problems will be passed on to the customer. Process cycle-time reductions, coupled with improvements in yield, have made statistical process control a valuable tool from both a cost reduction and a customer satisfaction standpoint. With its emphasis on early detection and prevention of problems, statistical process control has a distinct advantage over quality methods, such as inspection, that apply resources to detecting and correcting problems in the end product or service. In addition to reducing waste, statistical process control can lead to a reduction in the time needed to produce the product or service from end to end. This is partly because the final product is less likely to need rework, but it also results from using statistical process control data to identify bottlenecks, wait times, and other sources of delays within the process.

2.1.2 Process variables

Flow, pressure, temperature and level are the four most common process variables. The control of these embodies many of the difficulties existing in all major process industries. Based on core process technology, worldwide process industries have delivered control solutions for some of the most intractable control problems. In most cases, there exists some general-purpose controller that can be used across industries to control a process variable directly without the need for system redesign (plug-and-play).

In this subsection, we present control solutions relating to temperature, pressure, flow, and level process controls with roadmaps for selecting appropriate controllers for specific control problems.

(1) Flow process control

Of the four variables mentioned flow is probably the least difficult to control but has the largest number of loops above, because it is a continuous process moving material from the beginning of the process to

its end. Typically, the flow loop is controlled by manipulating valves, a variable frequency drive (VFD), or a pump.

A flow loop is naturally a first-order non-linear process with a small delay time. Since flow loops are the foundation of a process control system (for instance, they are typically used as the inner loop of a cascade control system), control of their performance is critical. How quickly the process variable stabilizes, whether there is an overshoot, how fast the overshoot damps out, and how fast the process variable can settle within a desired range are all quality criteria for the flow loop. In addition, inevitable wear and tear on the actuator can increase non-linearity in behavior. Therefore, the challenge for flow control is how to control a nonlinear process with stringent control performance requirements.

In flow process control, the commonly used actuators are nonlinear components by nature, including:

(a) a control valve, which is almost never a linear component; it can have a concave, convex, or square-shaped nonlinear relationship between its input and output; some of them even have hysteresis behavior that makes the problem much worse;
(b) a variable-frequency drive (VFD), which saves energy but is naturally a nonlinear device;
(c) a flow pump driven by a pulse-width modulator (PWM) based on the duty cycles, which does not necessarily have a linear relationship with the flow.

An example of flow process control is a gas-mixing process. For instance, if an iron and steel complex, operating units including blast furnaces, oxygen furnaces, and coking ovens all produce gases as by-products. Many plants discharge these gases into the atmosphere, wasting valuable energy and causing severe air pollution. A gas plant mixes these gases to produce fuel for the furnaces in metal casting and rolling mills. The quality of the mixed gas is measured by its heating value. Gases with inconsistent heating value can cause major control, quality, and production problems due to over- or under-heating. Even during normal production, gas supply and demand can change randomly. Major operating units such as blast furnaces and reheating furnaces may go online and offline periodically, causing huge disturbances in gas flow, pressure and the heating value. Online heating value analyzers are available, but are not usually used during normal operations, as they are difficult to maintain and overly expensive. Flow process control is applied to this gas-mixing process to monitor and control the gas heating value automatically in all operating conditions. For this purpose, a turnkey solution is used for heating value measurement and control. Using special, soft-sensor technology, the heating value can be accurately calculated online. An offline heating value analyzer can be used to calibrate the calculated value. Using model-free adaptive (MFA) controllers, gas flow and differential pressure loops can be effectively controlled. Robust MFA controllers are also used to handle the constraints and protect the system from running in vicious cycles. An MFA controller is used to control the gas heating value by cascading the gas flow and pressure controllers.

(2) Pressure process control

Pressure is another key process variable since its level is critical for boiling, chemical reaction, distillation, extrusion, vacuuming, and air conditioning. Poor pressure control can cause major safety, quality, and productivity problems. Overly high pressure inside a sealed vessel can cause an explosion. Therefore, it is highly desirable to keep pressure well controlled and maintained within safety

limits. There are a number of reasons why a pressure loop is difficult to control, including the following.

(a) A pressure loop is nonlinear, thus a propotional-integrate-differentiate (PID) or model-based controller may work well in its linear range, but fail in its nonlinear rage. A natural gas pipeline and a fluidized-bed boiler are two examples of such a pressure loop.

(b) The pressure process is a multivariable process that cannot be efficiently controlled by using single-input-single-output controllers due to interactions among the variables. For example, multiple gas pipelines may draw gas from a master pipeline. Therefore, when the load changes, these pipelines will interact with each other.

(c) Some pressure processes can generate high-speed and open-loop oscillations due to the poor frequency domain behavior of this process. For example, the pressure field and Mach speed value of an ultrasonic wind-tunnel, widely used in aerospace simulations, is open-loop oscillating. In this simulation experiment, it is very difficult to get an open-loop oscillation under control.

(d) The pressure process often involves large and varying time-delays. For example, pressure in municipal gas grids or a product powder transport system has large and varying time delays. Unfortunately, a PID controller cannot effectively control a process with large and varying time delays so more complex solutions must be sought.

(e) Pressure loops are typically noisy; in other words the measured pressure may jump up and down due to the nature of the pressure loop and the pressure sensor used.

In applications, use of model-free adaptive (MFA) controllers can help us to deal with some problems in pressure process control. Nonlinear MFA controllers can control nonlinear processes with no nonlinear characterization required. Robust MFA forces the pressure to stay inside the desired boundary of nonlinear control. The interactions between multivariable zones can be decoupled, by means of multiple-input multiple-output MFA control strategies, or algorithms, or feedback and feedforward MFA controllers. After choosing a phase-angle during configuration with a high-speed flex-phase MFA controller or a nonlinear flex-phase MFA controller, the flex-phase MFA can effectively control processes with bad behavior in the frequency domain. Furthermore, anti-delay MFA can effectively control processes with large time delays. Time-varying MFA can control processes with large and varying time delays. Low-pass filters can be used to screen the high-frequency noise. Since the MFA controller is not noise-sensitive, a filter may not be required unless the signal-to-noise ratio is so high that the control performance is obviously affected. Since the pressure loop requires fast sample and control update rates, the personal computer (PC)-based MFA control system may not be fast enough to control pressure loops. Embedded MFA control products or dedicated I/O cards in the PC will provide sufficient sample rates for pressure control. All embedded MFA control products can be used for pressure control. Some MFA control products have a user-entry field to pinpoint when the process variable is noisy. Based on the core MFA control method, various controllers have been developed to solve specific control problems.

A typical example of a pressure process is an onshore or offshore three-phase oil separator train, consisting of two separator vessels, with or without a slug catcher. Crude oil consisting of water, oil, and gas is separated by the system. Each vessel is typically controlled as a standalone unit. There are instances where a trip due to high pressure has occurred in one of the vessels while there is still spare capacity in another. Also, the gas pressure loop is nonlinear in nature and a random pulse of

high pressure can cause the pressure loop to swing, resulting in frequent ignition of excess gas. This is a combined control and optimization problem, requiring an effective solution in order to control critical level and pressure loops and maximizing the separation capacity. A multiple-input multiple-output MFA controller is used to control the oil levels in a multivariable fashion. It can coordinate the related oil levels by simultaneously manipulating the related valves to prevent too high an oil level in one vessel while there is still storage room in others. This special MFA can balance the oil levels for both vessels, and so maximize usage of the vessel capacity. Since no process models are needed, commissioning and maintenance of the system is simple, with guaranteed performance. In another aspect, nonlinear MFA controllers are used for pressure loops. They easily configured to deal with gas loop nonlinearity and slug problems since nonlinear characterization is not required.

(3) Temperature process control

Temperature is important in industrial processes because it its value is critical for combustion, chemical reaction, fermentation, drying, calcination, distillation, concentration, extrusion, crystallization, and air conditioning. Poor temperature control can cause major safety, quality, and productivity problems but it can be difficult to control in some applications. The many reasons why temperature is difficult to control are described in the following.

(a) Most temperature processes are very nonlinear because the dead band and slip-stick action of all the control valves make the temperature loop nonlinear. This means a PID or model-based controller may work well in its linear range and fail in its nonlinear range. Temperature processes are always looped at high speed due to ramping temperature up and down across a wide range at high speed. This is especially serious in the case of rapid thermal processing units for wafer treatment or for thermal testing of materials.

(b) In industry, it is often much faster to add heat to an operating unit than to take the heat away, if cooling is not available. Thus, the time constant can vary dramatically depending on whether the temperature is going up or down. Temperature processes are therefore slow and often time-varying. Varying time delays and time constants can easily cause a PID to oscillate or become sluggish. The PID can be tuned for certain operating conditions, but may fail when the process dynamics change.

(c) When subjected to large load changes and large inflow changes, temperature processes often deviate from normal. For example, steam generators in co-generation plants have to deal with large steam load fluctuations due to variations in steam users' operating conditions; e.g. tomato hot-breaks for tomato paste production: tomatoes are dumped in by the truck-load, causing significant inflow variations. For large load changes, if the load doubles, it requires twice the amount of heat to maintain the temperature. Feedforward control is often required. For large inflow changes, if the inflow is solid, it is difficult to measure the flow rate; so feedforward control is not a viable solution.

(d) Some temperature processes are multiple-input single-output processes; however, some temperature processes are single-input multiple-output processes. For example, an air-handling unit of a building control system manipulates the heating valve, the cooling valve, and the damper, based on split-range control. This is a multiple-input single-output process. In distillation columns, both the bottom temperature and the tray temperature need to be controlled, but the reboiler steam flow is the only manipulated variable. It forms a single-input multiple-output process. In such a process, one controller has to deal with multiple processes

such as heating and cooling. A fixed controller like a PID needs to be re-tuned when the control mode changes. In a single-input multiple-output process, the controller has only one variable to manipulate, but needs to control or maintain two or more process variables. In such cases, single-loop controllers such as a PID are not sufficient.

Based on the core MFA control method, various MFA controllers have been developed to solve specific control problems involving temperature. Without having to build on-line or off-line process models, an appropriate MFA controller can be selected and configured to control a complex temperature loop. MFA provides easy and effective solutions for previously intractable temperature control problems.

An example of temperature process control is the MFA control of multi-zone temperature loops. For instance, in a delayed coking process temperature loops can be controlled in multiple zones. In this control system, a cooker consists of two coking furnaces, each having two combustion chambers. High temperatures create carbon that clogs pipes, and a below-specified temperature causes an insufficient reaction so that the yield drops. Control difficulties result from large time delays; serious coupling between loops because the separation wall between the two chambers is quite low; multiple disturbances in gas pressure, oil flow rate, oil inflow temperature and oil composition. The oil outlet temperature is sensitive to gas flow rate change, and the temperature specification is tight ($\pm 1^{\circ}$C). An MFA control system running on a PC was networked to the existing distributed control system. The original cascade control design was simplified to eliminate disturbances and uncertainties. The new system regulates fuel flow directly to control the oil outlet temperature. A 2×2 anti-delay MFA controller on each furnace solves the large time delay and coupling problems. MFA controllers compensate for disturbances and uncertainties. Constraints on controller outputs prevent temperatures running too high or too low.

(4) Level process control

In process industries, the importance of level control is often overlooked. Typically, levels are controlled by manipulating the inflow or outflow to or from the operating unit, and it is considered to be an easy loop to control. Actually, it is difficult to tune a PID controller to give good control performance under all conditions, due to potential inflow and outflow variations of the operating unit. Overly tight level control will result in too much movement of the flow loop, which can cause excessive disturbances to the downstream operating unit. Thus, the PID level controller is usually detuned to allow the level process to fluctuate; so the variations of the outflow are minimized. The detuned PID, however, cannot provide rapid control of large disturbances, which may result in safety problems during a plant upset. In addition, oscillations in level during a process can cause the process to swing, which also results in a lower yield. Robust MFA controllers have been used to control level processes, which result in smooth material and energy transfers between the operating units, and also protect the levels from overflowing or becoming too dry during abnormal conditions.

To be successfully used for level control, a robust MFA controller must be configured in advance, considering the following configuration parameters:

(a) Upper and lower bound

These are the bounds within which the process variable is to be kept. These are “intelligent” upper and lower boundaries which typically are the marginal values which the process variable should not

exceed. The process variable is unlike the controller output where a hard limit or constraint can be set. The process variable can only be changed by manipulating the output. Thus, the upper and lower bounds for the process variable are very different from the output constraints.

(b) Gain ratio

This is coefficient ratio that indicates how much the process variable should be multiplied by to determine the level of MFA action. A value of three is common in which case, as the process variable approaches its bounds, the MFA will treat as if it had received a value three times the actual. Notice that this is not a gain scheduling approach, although it appears to be. Gain scheduling is not able to resolve the complex problems described.

Based on the core MFA control method, various MFA controllers have been developed to solve specific control problems. This method applies to level process control as well. The robust MFA controller is well suited to control conventional level loops and provides good protection from running too high or too low.

Multi-input multi-output MFA can control the density and level for operating units such as evaporators. Two examples are introduced below to illustrate how to control level processes:

(a) Boiler steam drum level process control

The level of steam in the drum needs to be kept near a middle value to prevent either heat stress on the boiler water tubes (level too low) or corrosion (level too high). Improper control of the level can cause system shutdown, waste of energy, and shorter equipment life. Key variables affecting the steam level are feed water inflow, steam outflow and fuel/mixed inflow. Each variable has its own distinctive type of disturbance. Cold feed water creates a time delay in the process. A sudden increase in steam outflow causes a distinctive shrink and swell response. This will confuse the controller because the process will act in a different manner temporarily. A three-element MFA control system can effectively control the steam level. The MFA level controller is cascaded with the feed-water controller to regulate the level, and compensate for disturbances from feed water and steam outflow. The anti-delay MFA controller handles the widely varying delay time. Feedforward MFA controllers keep the feed water supply in balance.

(b) Evaporator level process control

Evaporators are operating units for dewatering, concentration, and crystallization processes in various industries. If the density or the consistency of the product is an important quality factor, the level process control will be much more difficult. For instance, tomato processing plants use evaporators to produce tomato paste. The density of the tomato paste is the most critical quality variable which needs to be controlled within a small range. The variables that can be manipulated are the inflow and outflow, which affect both density and fluid level at the same time. The evaporator, by its nature, is a multi-input multi-output process. When using single-loop controllers to control the level and density separately, both loops can oppose each other, causing major problems. A multivariable MFA controller can control density and level, quickly and tightly by manipulating inflow and outflow simultaneously. An MFA constraint controller protects the evaporator level from running too high or too low.

2.1.3 Control methodologies

Industrial process control strategies are used for controlling a process variable that fluctuates around a set point. The industrial process controller has a differentiator which determines a difference signal between a process variable and a set point. The subsection below will very briefly introduce several advanced process control methods for dealing with such situations of including open and closed loops, adaptive control, PID control, robust control, predictive control, and optimum control methods.

(1) Open and closed loops

Although open loop and closed loop are important methods of process control, the two terms are often not clearly distinguished. The differences between open-loop control and closed-loop are demonstrated in the following two examples; of a computer numerical control (CNC) system and a room temperature control system.

(a) CNC systems

A CNC system requires motor drives to control both the position and the velocity of machine axes. Each axis must be driven separately, and must follow the command signal generated by the numerical control. There are two ways to activate the servo drives; the open-loop system and the closed-loop system.

In an open-loop CNC system, programmed instructions are fed into the controller through an input device. These instructions are then converted to electrical signals by the controller and sent to the servo amplifier to drive the servo motors. The cumulative number of electrical pulses determines the distance each servo drive will move, and the signal frequency determines the velocity of movement. The primary characteristic of the open-loop system is that there is no feedback system to check whether the desired position and velocity has been achieved. If system performance has been affected by load, temperature, humidity, or lubrication, then the actual output could deviate from that desired. For these reasons, the open-loop CNC system is generally used in point-to-point systems where accuracy is not critical. Very few, if any, continuous-path systems utilize open-loop control. Figure 2.2 illustrates the control mechanism of an open-loop CNC system.

The closed-loop CNC system has a feedback subsystem to monitor the actual output and correct any discrepancy from the programmed input. This can be either analog or digital. Analog systems measure the variation of physical variables, such as position and velocity, as voltages. Digital systems monitor output variations by means of electrical pulses. Closed-loop systems are very powerful and accurate because they are capable of monitoring operating conditions through feedback subsystems and can compensate for any variations automatically in real time. Most modern closed-loop CNC systems are able to provide very close resolution of 0.0001 of an inch. Closed-looped systems would, naturally, require more control devices and circuitry in order to implement both position and velocity control. This makes them more complex and more expensive than open-loop systems. A closed-loop CNC system is shown in Figure 2.3.

(b) A room heating system

In the case of open-loop control of the room temperature (θ_R) according to Figure 2.4, the outdoor temperature (θ_A) will be measured by a temperature sensor and fed into a control device. In the case of

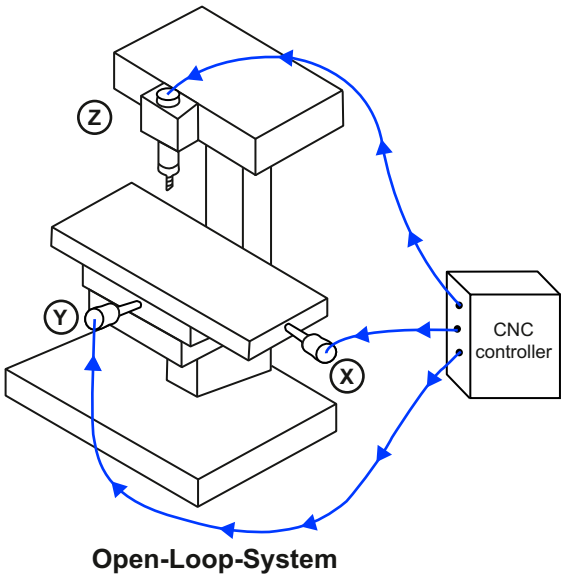


FIGURE 2.2

The open-loop CNC system.

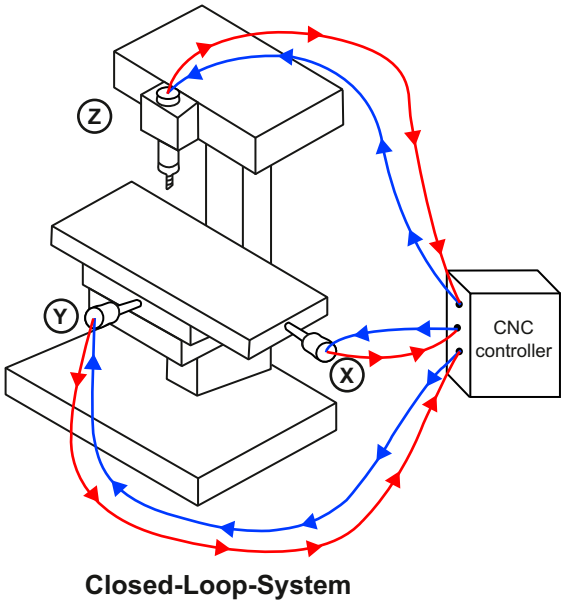


FIGURE 2.3

The closed-loop CNC system.

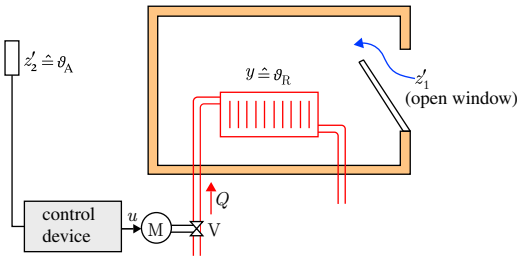


FIGURE 2.4

Open-loop control of a room heating system.

changes in the outdoor temperature θ_A (= disturbance z'_2) the control device adjusts the heating flow Q according to a slope function given by $Q = f(\theta_A)$ using the motor M and the valve V. The slope given by this function can be tuned in the control device. If the room temperature θ_R is changed by opening a window (= disturbance z'_1), this will not influence the position of the valve, because only the outdoor temperature will influence the heating flow. Hence, this control principle will not compensate for the effects of all disturbances.

In the case of closed-loop control of the room temperature, as shown in Figure 2.5, the room temperature θ_R is measured and compared with a set-point value ω, (for example, $\omega = 20°C$). If the room temperature deviates from the given set-point value, a controller indicated by C alters the heat flow Q. All changes of the room temperature θ_R, including those that are for example, caused by opening the window or by solar radiation, are detected by the controller and compensated for.

Block diagrams of open-loop and closed-loop temperature control systems are shown in Figures 2.6 and 2.7, and show the difference between the two control types very clearly. To this point, we can define systems in which the output quantity has no effect upon the process input quantity as open-loop control systems, and systems in which the output affects the process input quantity in such a manner as to maintain the desired output value are called closed-loop control system.

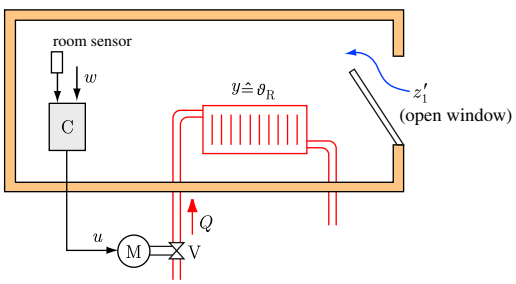


FIGURE 2.5

Closed-loop control of a room heating system.

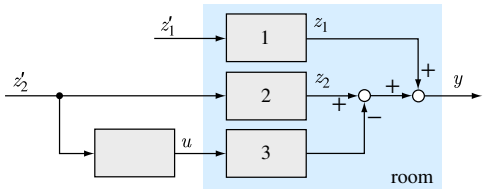


FIGURE 2.6

Block diagram of the open-loop control of the room heating system.

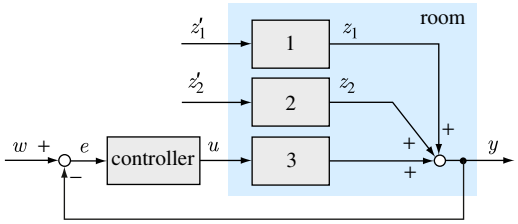


FIGURE 2.7

Block diagram of the closed-loop control of the room heating system.

(2) Adaptive control

An adaptive control system can be defined as a feedback control system intelligent enough to adjust its characteristics in a changing environment so that some specified criteria are satisfied. Generally speaking, adaptive control methods are mainly suitable for (a) mechanical systems that do not have significant time delays; and (b) systems that have been designed so that their dynamics are well understood.

Adaptive control methods, either model reference or self-tuning, usually require some kind of identification of the process dynamics. This contributes to a number of fundamental problems such as (a) the amount of offline training required, (b) the trade-off between the persistent excitation of signals for correct identification and the steady system response for control performance, (c) the assumption of the process structure, and (d) the model convergence and system stability issues in real applications. In addition, traditional adaptive control methods assume knowledge of the process structure. They have major difficulties in dealing with nonlinear, structure-variant, or with large timedelays processes.

(3) PID control

Many industrial processes are still controlled manually, or by 60-year-old PID controllers. PID is a simple general-purpose automatic controller that is useful for controlling simple processes. However, PID has significant limitations:

(a) PID works in process that is basically linear and time-invariant; it cannot control effectively complex processes that are nonlinear, time-variant, coupled, or have large time delays, major disturbances, and uncertainties. PID is not adequate for industrial processes with changing fuels and operating conditions.

(b) PID parameters have to be tuned properly. If the process dynamics vary due to fuel changes or load changes, PID needs to be re-tuned. This is often a frustrating and time-consuming experience.
(c) PID is a fixed controller, which cannot be used as the core of a smart control system.

(4) Robust control

Robust control is a controller design method that focuses on the reliability (robustness) of the control algorithm. Robustness is usually defined as the minimum requirement a control system has to satisfy in order to be useful in a practical environment. Once the controller is designed, its parameters do not change and control performance is guaranteed. The design of a robust control system is typically based on the worst-case scenario, so that the system usually does not work at optimal status under normal circumstances.

Robust control methods are well suited to applications where system stability and reliability are top priorities, where process dynamics are known, and variation ranges for uncertainties can be estimated. Aircraft and spacecraft controls are some examples of these systems.

In process control applications, some control systems can be designed with robust control methods, especially for those processes that are mission-critical and have large uncertainty ranges, and small stability margins.

(5) Predictive control

Predictive control, or model predictive control (MPC), is one of only a few advanced control methods that are used successfully in industrial control applications. The essence of predictive control is based on three key elements; (a) a predictive model, (b) optimization in range of a temporal window, and (c) feedback correction. These three steps are usually carried continuously by online out programs.

Predictive control is a control algorithm based on a predictive model of the process. The model is used to predict the future output based on historical information about the process, as well as anticipated future input. It emphasizes the function of the model, not the structure of the model. Therefore, a state equation, transfer function, or even a step or impulse response can be used as the predictive model. The predictive model is capable of showing the future behavior of the system. Therefore, the designer can experiment with different control laws to see the resulting system output, using computer simulation.

Predictive control is an algorithm of optimal control. It calculates future control action based on a penalty or a performance function. The optimization of predictive control is limited to a moving time interval and is carried on continuously on-line. The moving time interval is sometimes called a temporal window. This is the key difference from traditional optimal control, which uses a performance function to judge global optimization. This idea works well for complex systems with dynamic changes and uncertainties, since there is no reason in this case to judge optimization performance over the full time range.

(6) Optimum control

Optimal control is an important component of modern control theory. In principle, optimal control problems belong to the calculus of variations. Pontryagin's maximum principle and Bellman's dynamic programming are two powerful tools that are used to solve closed-set constrained variation problems, which are related to most optimal control problems. The statement of a typical optimal

control problem can be expressed as follows: The state equation and initial condition of the system to be controlled are given. The objective set is also provided. Find a feasible control, such that the system that starts from the given initial condition transfers its state to the objective set, and in so doing minimizes a performance index.

In industrial systems, there are some situations where optimal control can be applied, such as the control of bacterial content in a bioengineering system. However, most process control problems are related to the control of flow, pressure, temperature, and level. They are not well suited to the application of optimal control techniques.

(7) Intelligent control

Intelligent control is another major field in modern control technology. There are different definitions regarding intelligent control, but it denotes a control paradigm that uses various artificial intelligence techniques, which may include the following methods: (a) learning control, (b) expert control, (c) fuzzy control, and (d) neural network control.

2.2 INDUSTRIAL MOTION CONTROLS

2.2.1 Motion control applications

In industry, there are many applications in which there is a need to command, control, and monitor the motion of those components, parts of machines, equipment, or systems. The desired motion profile of such devices must sometimes be changed either during normal operation, at set-up, or under emergency conditions. This is known as industrial motion control. Industrial motion control primarily delivers control functions for velocity, acceleration and deceleration, position or torque.

(1) Velocity control

Velocity, or speed control is a common motion control mechanism. Several aspects of the velocity behavior of the system to be controlled must be clarified at the outset. Firstly, what is the speed required to operate the application? Secondly, does load of this application vary with speed or is load constant?

For a typical robot, differential velocities are used for steering. For example, in order to go straight, both motor shafts have to be turning at precisely the same rate, and the wheels must also be the same diameter. In another example, a machine tool axis will require, in general, constant thrust over a fairly wide range of cutting speeds, plus have a high-speed requirement at low load for rapid traversing. This results in an overall speed range of two or three orders of magnitude. In contrast, a machine tool spindle that drives a part in a lathe, or tool for milling, will require fairly constant power over a speed range of perhaps 5 to 1 as supplied by the motor. This is because the transmissions are generally added to further extend the constant power range.

Another consideration of velocity control is speed regulation. Speed regulation is generally expressed as percent of speed. Speed regulation may be short-term or long-term dependin on the application. Short-term regulation would be necessary for speed deviation due to some transient load of a known quantity. Long-term regulation would be needed for speed control over seconds, minutes, or longer. In addition, speed ripple in a system, often the result of motor and driver design, may be a concern at certain frequencies to which the application is sensitive. Examples of this would be the

effects of speed ripple on the surface finish of parts made by machine tools, or on the coating consistency of metals in an optical plating process driven by motors.

(2) Acceleration and deceleration control

The acceleration and deceleration controls is a higher level operation than velocity control in motion control. The acceleration or deceleration rate will affect the forces in the system, since torque is the product of inertia and rate of changed speed. It is important to include the inertia of the actuator or motor in any force calculation of this kind because its inertia may contribute considerably to the torque required. The selection of acceleration or deceleration profiles will also affect control performance. Linear acceleration is needed for a motor to change smoothly from one velocity to the next until the target speed is achieved. However, acceleration that follows a non-linear, curved profile gives rates of change that vary with both position and velocity.

Curve-profiling is important in acceleration and deceleration control. S-curve acceleration has a low initial rate of change and then increases to a maximum rate, then decreases again until the target speed is achieved. A parabolic profile begins acceleration at a high rate, and then decreases. Braking a car to a stop at exactly the right spot is one example of deceleration control using a parabolic profile.

(3) Position control

Position control entails the control of motion of displacement, which is the change of position with respect to time. This includes command, control, and the monitoring of motion. This can be as simple as the change in velocity obtained by limit switches on a simple slide drive, or as complex as linear and circular interpolation among axes on a multi-axis machine. Position control typically needs to be able to change certain parameters of the required motion flexibly. For example, the length of the move or the speed of the system may change based on variables in the process, or the parts being manufactured. The resolution of the position control, i.e., the smallest unit of displacement, needs to be defined. Along with the resolution, the accuracy and repeatability of the motion displacement need to be determined. Resolution, accuracy, and repeatability are common performance measures associated with position feedback devices such as encoders and resolvers; but the specification of a complete motion system must also take into account the mechanical system and position controller.

Position control loops are performed by a user inputting a desired position and the servo logic drives the output shaft to that position with whatever force is required. Normally one does not think of position control as being appropriate for doing motion control on small robots, but, in fact, it is a very good technique. The trick is to profile a series of positions for the robot to be in, feeding that string of positions to the servo on a periodic basis. This is called position-profiling. With position-profiling the forward velocity of the robot can be any arbitrary low amount. It might be that the motor shafts advance only every so often, but they will advance smoothly and the robot will crawl along. With profiling, acceleration becomes easy to do as well.

(4) Torque control

Torque control is widely used as a method for controlling the torque or force in a system, independently to its speed. An example would be a simple feed or take-up tension control in making rolls. Maintaining constant tension by varying torque at the rolls as a function of roll diameter will certainly result in a constant power requirement. A more complex tension control might require a changing or tapered tension as a function of roll diameter. As in the evaluation of a velocity controlled system,

a torque controlled system needs to be quantified by a number of parameters: what is the required torque range? Over what speed range must the torque be provided? Is torque ripple of concern, and, if so, what frequencies of ripple present a problem?

Torque control is a unique method for controlling AC motors. In pulse-width modulation drives, the output frequency and voltage are the primary control reference signals for the power switches, rather than the desired torque in and out of the motor shaft. The torque control principle can be illustrated via this mechanical analogy of the continuous calculation of the best angle at which to rotate a shaft, with a given arm length and the forces available. These electrical force vectors are generated with the help of semiconductor switches called integrated gate bipolar transistors (IGBT). Testing of rotating machines such as gears, engines, and complete cars is a demanding task. High accuracy and dynamic load control, that is, control of torque, are both needed for clutch transmissions being introduced into the current generation of automobiles. AC motors drive these test rigs. When manufacturing such test rigs, which have stringent performance requirements, careful consideration must be given to the AC machines and drives to be used (control of speed and torque are paramount). The way the AC motor is controlled by its drive has a primary effect on these considerations.

Advanced motion control

Advanced motion control requires two control mechanisms: servo and feedback. In most industrial applications, these two mechanisms are combined to perform servo-feedback functions for motion control.

(1) Servo control

The basic reasons for using servo systems in industrial motion control include the need to improve transient response times, to reduce steady-state errors, and to reduce the sensitivity to load parameters. Improving the transient response time generally means increasing the system bandwidth. Faster response times mean quicker settling, allowing for higher machine throughput. Reducing the steady-state errors relates to servo system accuracy. Finally, reducing the sensitivity to load parameters means the servo system can tolerate fluctuations in both input and output parameters.

Servo control in general can be broken down into two fundamental classes of problem. The first deals with command tracking i.e., how well the actual motion follows what is being commanded.

The second general class of servo control addresses the disturbance rejection characteristics of the system. The typical commands of rotary motion control are position, velocity, acceleration and torque. For linear motion, force is used instead of torque. The part of servo control that deals with this directly is often referred to as feedforward control. Feedforward control predicts the internal commands needed for zero errors.

The feed forward commands can be thought of as the internal commands that are needed such that the user's motion commands are followed without any error, assuming of course that a sufficiently accurate model of both the motor and load. The disturbance rejection control reacts to unknown disturbances and modeling errors. Disturbances can be anything from torque disturbances on the motor shaft to incorrect motor parameter estimations used in the feedforward control. The familiar proportional-integral-derivative (PID) and proportional-integral-velocity (PIV) controls are used to combat these types of problems.

Complete servo control systems combine both feed forward and disturbance rejection control to optimize overall performance.

In a PID control system, the PID controller calculation involves three separate parameters; the proportional, the integral and the derivative values. The proportional value determines the reaction to the current error, the integral determines the reaction based on the sum of recent errors, and the derivative determines the reaction to the rate at which the error has been changing. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve, or the power supply of a heating element. By tuning the three constants in the PID controller algorithm, the PID can provide a control action designed for specific process requirements. The response of the controller can be described in terms of responsiveness to an error, the degree to which the it overshoots the set-point and the degree of system oscillation. Note that the use of the PID algorithm does not guarantee optimum control of the system, or system stability.

In contrast to the PID control, the PIV control requires knowledge of the motor velocity, labeled the velocity estimator. This is usually done by a simple filter, but significant delays can result, and must be dealt with if truly accurate responses are needed. Alternatively, the velocity can be obtained by use of a velocity observer. This observer uses other state variables in order to provide zero-lag filtering properties. In either case, a clean velocity signal must be provided for PIV control. To tune such a system, only two control parameters are needed the bandwidth and the damping ratio. An estimate of the motor's total inertia and damping is also required at set-up, and is obtained using the motor/drive set up utilities.

(2) Feedback control

Feedback can be described as a function that transforms inputs to outputs, such as an amplifier that accepts a signal from a sensor and amplifies it, or a mechanical gearbox with an input and output shaft.

When analyzing such systems we will often use transform functions that describe a system as a ratio of output to input.

There are two main types of feedback control systems: negative and positive. In a positive feedback control system the set-point and output values are added. In a negative feedback control the set-point and output values are subtracted. As a rule negative feedback systems are more stable than positive feedback systems. Negative feedback also makes systems more immune to random variations in component values and inputs.

2.2.2 Motion control systems

A motion system can be very complex and may include many different types of motion components. An example of such a system is a computer integrated manufacturing (CIM) system which receives as input a computer-aided design (CAD) data file. It inspects and loads tools into a manufacturing cell, makes a part in accordance with the CAD information, provides real-time adjustment the manufacturing process, and then collects, processes, and stores information for statistical process control purposes. This system will include many types of motion control systems; controllers, amplifiers, motors and actuators, and feedback devices. The combination of these components required to perform a given application will vary, and many considerations affect which type of system is the best one.

Often two factors need to be considered in designing a motion control system.

(1) At the top of a motion control hierarchy is the profile generator, which will typically contain many defined profiles. When a specific profile is selected, the profile generator will feed its velocity and position commands to the next block, which is the real-time control loop of the drive.

(2) The drive module can include both velocity and torque control inner loops or not, depending on the control technique used. The output of this block is drive power to the motor, with feedback from the motor output providing position information (typically through encoders or resolvers) plus velocity information (with tachometers) for the drive loop.

Figure 2.8 shows a block diagram of a typical motion control system that includes; the operator interface, usually a human machine interface (HMI) for communication between operator and controller; the motion controller, which acts as the brain of the system by taking the desired target positions and motion profiles, and creating the trajectories for the motors by outputting a voltage signal for motor or actuator to follow; application software, which gives target positions and motion control profiles commands; amplifiers (or drives), which take the commands in voltage signals from the controller and then generate the current required to drive or turn the motor; motors or actuators, which turn electrical energy into mechanical energy and produce the torque required to move to the desired target position; mechanical elements, which are designed to provide torque to some devices such as linear slides, robot arms, special actuators and so on; a feedback device, usually a quadrature encoder, which senses the motor position and reports the result back to the controller. A feedback function would not be required for some motion control applications such as controlling stepper motors, but is vital for servo motors.

2.2.3 Motion control technologies

Industrial motion control involves very complicated electronic and mechanical technologies. When implementing and testing a motion control system, we need to take account of many technical issues. The following lists some key issues for motion control technology.

(1) Repeatability and accuracy

It will involve a combination of art, science, and experience to choose the correct motion control components for a successful servo positioning system. Motion control systems typically employ

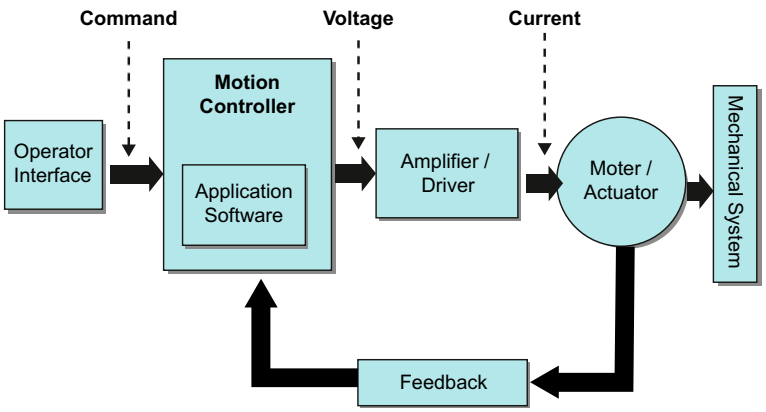


FIGURE 2.8

Block diagram of a motion control system.

a combination of belts, ball screws, lead screws, and motors for determining speed, torque, and direction. All of these four components are contained in the motion controller.

When selecting the components for a motion control system, the first factors to consider are speed and torque, which will determine whether the system should host a stepper motor or a servo motor. Steppers are usually superior for systems that operate at speeds lower than 1,000 rpm (rotations per minute) and less than 200 watts. By comparison, as shown in Figure 2.9, servomotors are preferred for speeds above 1,000 rpm and power levels above 200 watts. Each has a unique set of parameters that contribute to its accuracy, resolution, and repeatability.

The next issue for selecting motion system components concerns feedback devices. Steppers do not require feedback; servo motors use feedback devices by definition. Servo systems require one or more feedback signals in simple or complex configurations, depending on the specific needs of the motion system. Feedback loops record position, velocity or speed, acceleration and deceleration. Sometimes, feedback also includes "jerk"; the first derivative of acceleration with respect to time.

The motor amplifier or drive (as illustrated in Figure 2.8) is the part of the motion control system that takes commands from the motion controller, in the form of analog voltage signals of low current, and converts them into high current signals that drive the motor. Motor drives come in many different

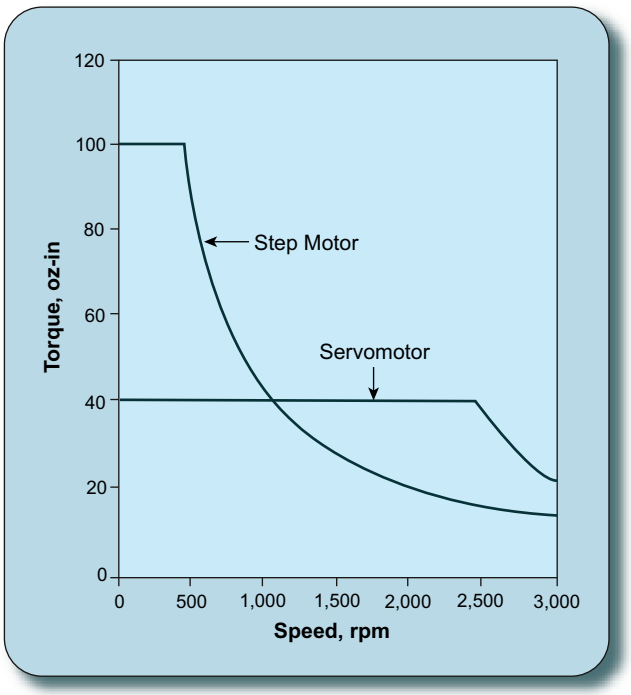


FIGURE 2.9

Two curves for torque against speed for a typical step motor and a typical servomotor of equivalent physical size and material content.

varieties and are matched to a specific type of motor. For example, a stepper motor drive connects to stepper motors, and not servo motors. Along with matching the motor technology, the drive must also provide the correct peak current, continuous current, and voltage to drive the motor. If a drive supplies too much current, it could risk damaging motor. If a drive supplies too little current, the motor does not reach full torque capacity. If system's voltage is too low, a motor cannot run at its full speed.

Motion control systems are usually specified to perform any one or a combination of three different types of moves. These include unidirectional, point-to-point bidirectional, and contouring modes. Unidirectional moves require point-to-point repeatable moves, in which the destination point is approached from only one direction. Accuracy for these systems is more difficult to achieve, as also serves as a measurement system. Highly repeatable unidirectional systems are usually also highly accurate. Bidirectional repeatability is more difficult to achieve because of backlash and hysteresis. Many motion controllers can compensate for repeatable gear backlash, but fail to handle other components with less predictable backlash. Supplying a secondary encoder for position information can compensate for backlash. Highly accurate and repeatable systems often use a position sensor outside the motor. Care must be taken with these systems since the hysteresis or dead-band would be enclosed within the position loop. Hysteresis is evident when the system is commanded to reach the same destination from opposite directions. A rotary encoder coupled to the motor would indicate that the load has reached its destination, but the actual position error is larger than the backlash alone. This hysteresis is caused by unseen clearances and elastic deformations. A linear encoder can compensate for backlash and hysteresis in a screw-driven positioning system. The hysteresis must still be minimized to avoid problems with oscillation, and systems with hysteresis potentials must minimized friction.

(2) Motor power transformation

In motion control systems, motors are the components that convert electrical power to useful mechanical power. Gearheads transform the mechanical rotary power to the desired combination of speed and torque if the motor cannot do so directly. Transmission systems are chosen on the basis of performance, size, cost, and various other constraints. Whilst each type of system has its own benefits and disadvantages, gearing systems hold many advantages for economic power consumption in small packages.

There are several types of electrical motors and power transmission devices. Depending on the application, a designer might first choose between alternating current (AC) and direct current (DC) motors, then between styles (including induction, universal, and permanent magnet), and finally between the many types specific to each style. The designer must also select a controlling device and amplifier to provide the proper input power for the motor. For example, the motor may be capable of supplying the necessary power, but not match the speed or torque requirements. Most motors are rated to operate for peak output around a certain continuous load range of torque and speed. To operate the motor outside that range would either under-utilize the motor and result in an oversized motor, or over-utilize the motor and potentially run the risk of damage motor. Although each type and style of motor operates differently, smaller motors typically operate more efficiently at higher speeds and lower torques. Larger motors operate at higher torques and lower speeds. This is obvious if the motor is viewed as a large moment arm.

High-energy, permanent magnet, servo motors have become extremely popular industrial motion controls. For these motors, a general rule of thumb is that if the length of the motor is doubled, then the

output torque capacity of the motor is also doubled. However, the output torque capacity of the motor will be quadrupled if the length is kept the same and the diameter of the motor is doubled. For maximum efficiency, proper sizing, and economy, it is best to design the motor for operation at, or slightly under, its designed load point of speed and torque.

The speed-reducing planetary gearhead as described adheres to the following:

ratio = $(N_s + N_r)/N_s$ (where N_s = number of sun gear teeth and N_r = number of ring gear teeth).

There are many variations of epicycle gearing that can be produced from this style. In some instances, the ring gear can be the driving gear or the output gear. Complex epicycle, or planetary gearing is also possible, in which gear clusters are used in planetary configurations. An automobile transmission is a good example of how several styles of planetary gearing can be utilized together.

To enhance the output of such a spur gear design, a style of involuted spur gearing, known as epicycle gearing (i.e., planetary gearing), was developed. Planetary gearing dramatically increased the strength of the gearhead while eliminating radial loading, by counterbalancing the effect of any one-gear engagement. Planetary gearheads distribute the input power coming from the sun gear (input pinion) to two or more planetary gears. Since the planet gears are positioned symmetrical to the pinion gear, radial loading on the pinion gear is eliminated. The planet gears are housed within a ring gear, which has gear teeth cut into the inside diameter in a reversed tooth profile. The separating forces (radial forces) produced from the planet gears' tooth engagement with the sun and ring gears cancel each other. The ring gear is stationary, and the input sun gear pinion drives the planet gears, which then walk in the same rotational direction as the sun gear.

Depending on the number of teeth in the sun gear and ring gear, each stage typically generates speed reduction ratios of between 3 and 10 times the input speed. A two-stage planetary gearhead, which has two 10-to-1 stages (i.e., 10 rotations of the input pinion results in one rotation of the output), yields 100 times the output torque (less efficiency) and 1/100 the rpm. If the gearhead were reversed so that the output shaft becomes the input shaft, the speed would increase by 100 times; the available output torque would decrease proportionately.

2.3 INDUSTRIAL PRODUCTION AUTOMATION

2.3.1 Definition and functions

The term industrial automation generally refers to the science and technology of process control and automatic machinery assembly in any industry. Industrial process control is discussed in section 2.1 of this textbook. This section discusses automatic machinery assembly, termed industrial production automation.

Industrial production automation is the use of computers, programmable numerical controllers and robots to complete machinery assembly in industry, particularly necessary advanced production systems. Such a production system will consist of a number of machines organized together to perform multiple operations. These machines function as manufacturing cells, which comprise all the operations required to manufacture a given product or group of products. Instead of being organized functionally all of the stamping equipment located in one specific area, for example manufacturing cells contain all of the machines and personnel necessary to create a product in one prescribed space, beginning with the raw materials and ending with the finished product. The

proximity of machines in a manufacturing cell cuts down on the waste that comes from having to transport materials from one place to another, and offers workers an opportunity to be cross-trained on a variety of machines. They are often used to streamline and automate material removal processes. There are many industrial robotic machines that can be integrated with deburring, grinding, and cutting machines to facilitate this. Material removal cells can also include ventilation parts and small fume removal cells. A small fume removal cell is often used to remove smoke or mist, or to collect dust. These manufacturing cells are also used in welding, sanding, and other high-speed machining processes.

There are three main flows that determine the organization of industrial production systems; material flow, energy flow, and information flow. The basic objective of industrial production automation is to identify the information flow, and to manipulate the material and energy flows of a given production system in the desired, optimal way. This is usually a compromise between economic, social and quality factors in order to obtain maximum benefits. The most common benefits of industrial production automation are:

(a) increasing productivity;
(b) reducing production costs;
(c) improving product quality and reliability;
(d) optimizing production flexibility and production schedules;
(e) enhancing production safety;
(f) controlling pollution of the environment due to production.

Productivity can be increased, either by better use of available production capabilities, for example by moving plant bottlenecks, or by production acceleration, i.e., by making the production process faster. When using computers, controllers and robots, production volume can be increased by running the production sequence up to the highest allowable limits of some critical variables, whilst continuously monitoring their actual values and their trends.

Production costs can be reduced by saving raw material, energy, labor, and by accelerating the process. Industrial production automation using computers, controllers and robots has two aims; saving labor in production, and reducing production time.

Product quality can be improved by better supervision of the production sequences, and improved engineering tools and conditions. On one hand, this is achieved by keeping the run-time conditions of the production system within tolerance limits. On the other hand, mathematical models of production processes allows online adjusting of relevant parameters when raw material properties or ambient conditions are changed. However, quality increase depends largely on the ability to measure the quality itself, which can be rather timing-consuming. Product reliability generally depends an product quality, so is generally thought of as a quality indicate.

Production flexibility is the major requirement for a system working under changing engineering conditions. Typical examples are batch-production, where frequent change between products is generally necessary, especially in the chemical, biochemical and petrochemical industries. The role of computer or controller in this case is to provide better production scheduling. Robots provide the automatic production. If computers, controllers and robots are combined properly, optimal utilization of available facilities and better control are readily enabled.

Production safety can be improved by production automation to prevent workers becoming victims of production accidents. In most industrial examples, the more automatic a production process is, the

fewer workers are required in the production process. Industrial production automation solves both production reliability and safety problems by exact control of process conditions; or in batch processes, by exact execution of the required successive steps, by vigilant process monitoring and by timely prediction of possible hazardous situations.

Industrial production automation is different from mechanization. In mechanization, human operators run the machinery. But in production automation, most of the work is facilitated by computers, controllers or robots. This raises social issues, particularly for populous countries. Industrial automation affects employment to a great extent unskilled labor, especially, since automated industries require only skilled laborers.

This does tend to have a cathartic effect on the working population. It forces people to take up skilled jobs, or makes them try to do so. Nowadays industrial production automation is very advanced, and has even replaced many skilled jobs. It is advancing rapidly throughout the world, and may in future affect more and more skilled jobs.

In spite of all these repercussions there is a greater emphasis on industrial production automation. Initially its main purpose was to maximize production, but now focus has a shifted to enhanced product quality. Industrial production automation has also made the workforce more flexible, and it is now quite feasible to switch over from one product to another.

Once the need for automation is understood, at least two questions should be asked. What are the aims of your automation? How can they be realized under the given engineering conditions? Sometimes, neither question can be fully answered. Automation aims are frequently difficult to define due to conflicting requirements; arising from complex energy, raw material, and product sale markets. Also, there may be several limitations within the production system itself. Even where the automation aims can be clearly defined, implementation decisions will have to be made based on economic factors and on the possible disruption of production during the installation of the automation process. There are three kinds of basic approach to industrial production automation.

(1) Partial production automation, which first removes the bottlenecks within a production system, and then gradually extends to complete production automation without total factory or plant or workshop shutdown: this is called a bottom-up approach.
(2) Complete production automation, usually suitable for the installation of new production systems, is generally known as the top-down approach.
(3) Integrated production automation functionally interconnects process control, supervisory, and commercial computers, which are available within the production system to increase production transparency, correctly plan strategy, and properly analyze markets.

Modern industrial production automation often involves networks on an enterprise scale, thus necessitating complicated engineering systems. Figure 2.10 is an illustration of such a network for an industrial production automation system. From this figure, we can get some insight into the complexity of industrial production automation systems.

2.3.2 Systems and components

Since the accelerated development in the twentieth century, industrial production automation has become a mature technology with wide application. The early system were off-line and closed-loop; the production processes and systems were completely separated from the operators. Later, these

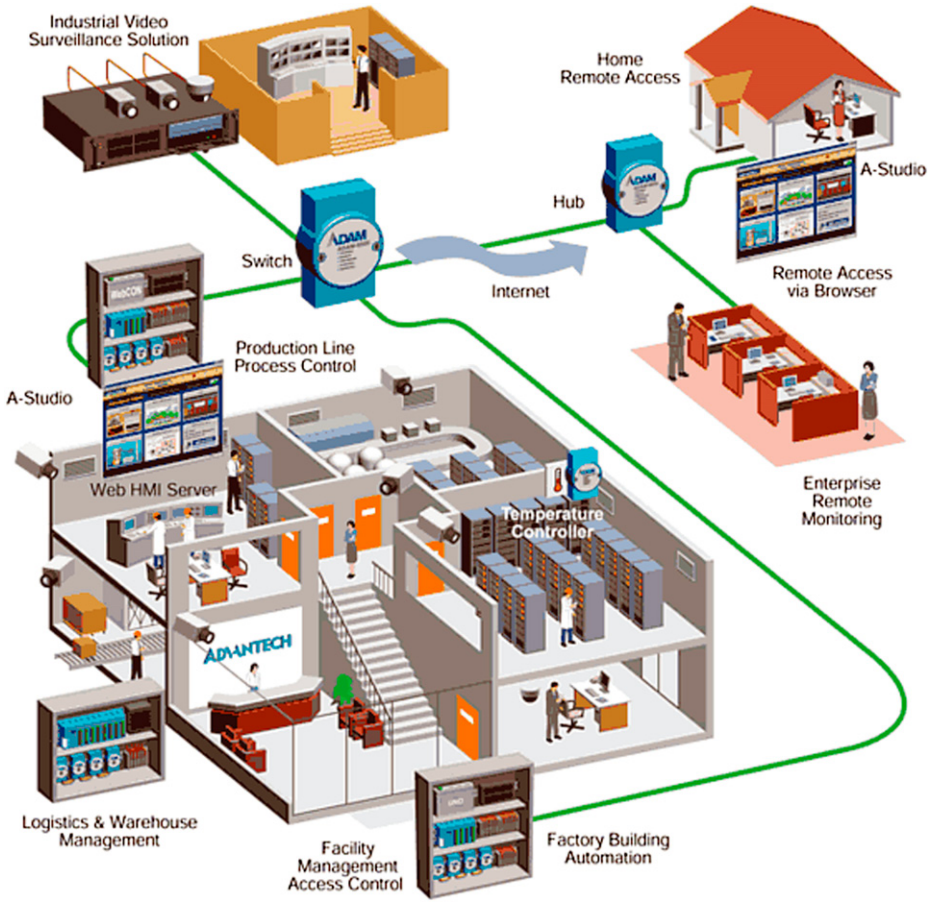


FIGURE 2.10

Network of a modern enterprise-scale production automation system.

(Courtesy of AdvanTech Co., Limited.)

off-line and closed-loop architectures were replaced by on-line and open-loop systems, in which operators interacted with production systems for monitoring and control purposes. Nowadays, modern industrial production automation can be characterized by two features:

(1) Computerized control technologies

The hardware of computerized technologies in industrial production automation are with numerical computers, intelligent and programmable controllers consisting of microprocessors and large-scale integrated circuits. The software elements are real-time operating systems and application-specific software systems. Computerized technologies offer digital and numerical capabilities which have

allowed industrial production automation to attain greater speed and flexibility, promoting many new applications.

(2) Distributed system architectures

The control elements of a distributed architecture that are most commonly used are on-line and open-loop technologies. The most common elements configuration are multi-agents and networks. Distributed industrial automation systems reside in open networks of remote interactions and communications. They give industrial production large-scale and remote production lines over factories and plants, by using networks.

Hierarchical Architecture

Although there are some different approaches, modern industrial production automation systems adopt the following four-layer hierarchical model. As shown in Figure 2.11, these four layers are the human machine interface (HMI) layer, the application layer, the system layer, and the production layer.

(1) HMI layer

There is an extensive mixture of human and machine labor in actual manufacturing and production systems. This situation, while desirable for its agility, poses some difficulties for efficient cooperation between humans and machines. The HMI consequently becomes very important to optimize the capabilities of installed production machines, in cooperation with their human operators. This layer captures all the abstractions in the development and operational phases of HMI modules, and is supported by open and commercially available platforms and tools. However, human interactions with these platforms and tools must be considered.

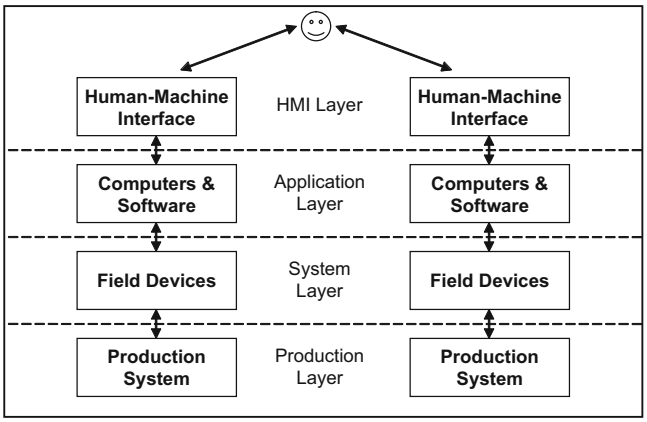


FIGURE 2.11

The four-layer hierarchical architecture of distributed industrial production automation systems.

(2) Application layer

This layer includes the software used in the analysis, design and implementation of applications. Based on the communication model, this layer allows an application program to ensures effective communication with other application programs in a distributed open-system. The application layer makes sure that the other party is identified and can be reached, authenticates either the message sender or receiver or both, ensures agreement at both ends about error recovery procedures, data integrity, and privacy, and determines protocol and data syntax rules at the application level.

(3) System layer

This layer includes the analysis, design and implementation of the system components. Implementations include the Fieldbus segments, the field devices, the microprocessors and other intelligent electronic units, and the network used for the real-time interconnection of Fieldbus segments. The design principles of this layer include the abstractions to support the assignment and distribution of the applications. These abstractions play the role of proxies of the actual real-world objects in the developer's workspace.

(4) Production layer

Real production lines, including boilers, valves, motors, etc., are the actual plant components that constitute the implementation objects of this layer. The design representations of the real-world objects form the design diagram of the controlled manufacturing process, usually drawn as pipe and instrumentation diagrams.

In additional to the four-layer hierarchical model, industrial automation experts also recommend a functional levels model, in which automation is divided vertically into four levels; management, scheduling, supervision, and direct control. As shown in Figure 2.12, at each level some automation functions are implemented, which operate on the next level down. For instance, the lowest automation level, the production direct control level, interacts through appropriate interfaces with the instrumentation on the production lines such as sensors and actuators. Its algorithms will be supplied by the next level up, i.e., by the production supervision level.

Components of a Production Automation System

Computerized and distributed production automation systems include these basic components:

(1) Hierarchical database

Industrial production automation requires dynamic large-scale information data at all its function levels. However, before the database was commonplace, data had to be stored in files. Early file management systems for industrial automation were designed to allow programmers to create, store, update and read files. In the earliest attempt to abstract the programs from the underlying physical data structures, indexed files were introduced to provide access to data via indexed keys. In early file management systems, a typical historic log file would be indexed by tag name, and data could be retrieved provided the tag name is known. Indexed files still had the limitations of a flat structure with a single point of access. For instance, a programmable logic controller (PLC) can manage data in a similar way, allowing programs to access the current value of a tag by reading a register within the

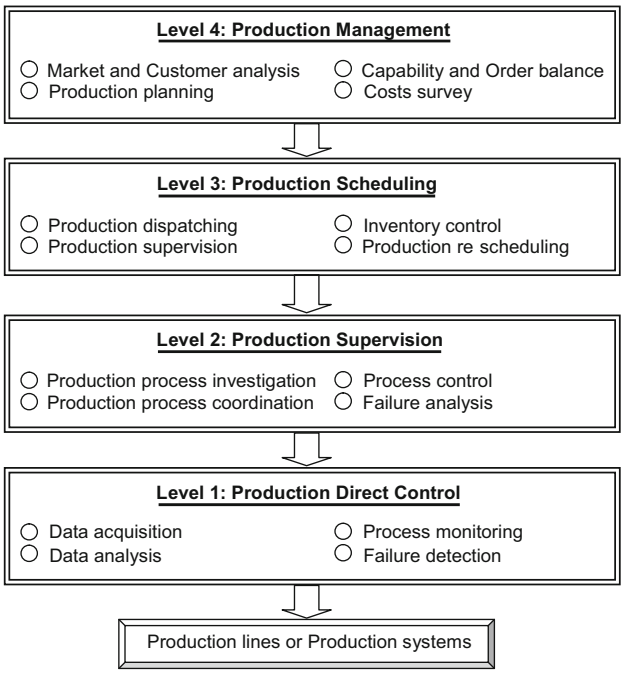


FIGURE 2.12

The functional levels of industrial production automation systems.

PLC. Programs need to know the exact location of data on the disk, and complex algorithms are required to compute these unique addresses.

Hierarchical databases were developed to address this limitation. These databases arrange data elements in strict hierarchies, where one category of data is a subset of another. For example, an industrial plant has several production lines, each production line has a number of machines, and each machine has many measurements (or tags) associated with it. With a hierarchical database, a query for data unrelated to the functional breakdown of the plant described above, but related to a specific product or batch or engineering unit, would be both complex and inefficient.

Since the world is not hierarchical, network databases that could accommodate more complex network relationships began to appear. Although programs were less dependent on physical data structures, they still needed to know how to navigate these structures, and were susceptible to any changes. Most of the production information management systems (PIMS) available today are based on hierarchical or network data models, and are subject to the limitations of these models. The network database is the best solution yet developed to minimize these limitations.

(2) Automation controllers and software

Embedded automation computers fulfil the needs of mission-critical automation applications. Their design, industrial features, embedded operating system (OS) and powerful computing technology

deliver reliability and flexibility. Automation controllers and their software include programmable automation controllers, distributed data acquisition and control systems, analog input/output modules, digital input/output modules, counter/frequency modules, communication modules, specific modules, and power supplies.

(3) Open HMI platforms

HMI platforms for automation needs include industrial panel PCs, industrial workstations, industrial monitors and touch panel computers, and so on.

(4) Industrial communication systems

Industrial communication series include industrial Ethernet switches, serial device servers, media converters, and multiple-port serial I/O cards. Advanced technology provides robust communication devices and cost-effective ways to manage network efficiency and other connected field devices.

2.3.3 Robotics and automation

Modern manufacturing set-ups rely increasingly on automation technology. Modern production systems are becoming more and more autonomous, requiring less operator intervention in normal operation. It is now common to have all equipment on the workshop floor connected by an industrial network to other factory resources. This requires use of computers for control and supervision of production systems, industrial networks and distributed software architectures. It also requires application software that is designed to be distributed on the workshop floor, taking advantage of the flexibility provided by using programmable controllers.

Production automation has posed problems, arising from the characteristics of the production pieces, e.g., non-flat objects with highly reflective surfaces, those very difficult to grasp and handle due to external configuration, heavy or fragile objects, pieces with extensive surfaces that are sensitive to damage, pieces with a high requirement for surface smoothness etc. Production set-ups for these types of products require high quality and low cycle times, since industries must maintain high production rates to remain competitive. Another restriction relates to the fact that most industries change products frequently due to trends, competition, etc. Lastly, there is the mixture of automatic and human labor production, which poses its own problems.

Robots with industrial software are key to solving the problems given above. In industry, robotics and automation are closely linked in the design and implementation of intelligent machines to carryout work too dirty, too dangerous, too precise or too tedious for humans. They also push the boundary of the level of intelligence and capability for many forms of autonomous or semi-autonomous machines. Robots are used in medicine, defense, space and underwater exploration, service industries, disaster relief, manufacturing and assembly and entertainment.

(1) Industrial robot types

Industrial robots are those numerical control systems that can be automatically controlled, and are reprogrammable, multipurpose manipulators that can move along in three or more axes. Some robots are programmed to carry out specific action repeatedly without variation, and with a high degree of accuracy. These actions are determined by programmed routines that specify the direction,

acceleration, velocity, deceleration, and distance of a series of coordinated motions. Other robots are much more flexible as to the orientation of the object; for example, for more precise guidance, robots are capable of machine vision sub-systems acting as their "eyes", linked to powerful computers or controllers. Artificial intelligence, or what passes for it, becomes an increasingly important factor in the modern industrial robot. Industrial robots are basically of the following types:

(a) articulated robots having arms with three rotary joints;
(b) Cartesian robots (rectangular coordinate robots, rectilinear robots) having three prismatic joints whose axes are coincident with a Cartesian (X, Y, and Z) coordinate system;
(c) gantry robots, which are a type of Cartesian robot that is suspended from an X or X/Y axis beam;
(d) cylindrical robots operate in a cylinder-shaped space or coordinate system and have at least one rotary joint and at least one prismatic joint;
(e) parallel robots such as hexapods having multiple arms, each of which has three concurrent prismatic joints; selectively compliant arm for robotic assembly robots are cylindrical and have two parallel joints to provide compliance in one selected plane;
(f) spherical robots have an arm with two rotary joints and one prismatic joint; the axes of a spherical robot form a polar coordinate system.

(2) Industrial robot applications

Industrial robots are used across a wide spectrum of industrial applications. Typical examples include:

(a) material handling; including pick and place, dispensing, palletizing, press tending, part transfer, machine loading, machine tending, and packaging, etc.;
(b) welding: including arc welding, spot welding, robot laser welding, resistance welding, plasma cutting, flux cored welding, and electron beams, etc.;
(c) miscellaneous: for instance grinding, drilling, thermal spraying, bonding and sealing, painting automation, deburring, robotic coating, robotic assembly, material removal, flame spray, polishing, etc.

Problems

1. Please explain what "industrial control engineering" is, and what its major activities are.
2. By analysis of the process control system given in Figure 2.1, explain how, in a continuous process control system, the process variables are monitored, and subsequent control actions are implemented to maintain variables within predetermined process constraints.
3. In Figures 2.2 and 2.3, which functions are keys to differentiate between an open loop CNC system and a closed loop CNC system?
4. Analyze the correspondence between Figures 2.4 and 2.6 for the open loop room temperature control; analyze the correspondence between Figures 2.5 and 2.7 for the closed loop room temperature control.
5. Do you agree with this conclusion: systems in which the output quantity has no effect upon the process input quantity are called open loop control systems; systems in which the output has an effect upon the process input quantity in such a manner as to maintain the desired output value are called closed loop control systems?
6. Please give an example of a motion control system in an industry. By analyzing this system based on Figure 2.9, specify the corresponding components in this system.
7. Design an automatic assembly line for the simplest washing machine; and plot the architectures of its hierarchical layers and function levels.

Further Reading

William S. Levine (Ed.). The Control Handbook. New York: CRC Press, 2000.
Dobrivoje Popovic, Vijay P. Bhatkar. Distributed Computer Control for Industrial Automation. New York: CRC Press, 1990.
CyboSoft (http://www.cybosoft.com). 2008. Process control solutions. http://www.cybosoft.com/solutions. Accessed: April 2008.
Christian Schmid. Open loop and closed loop. 2005 05 09. http://www.atp.ruhr uni bochum.de/rt1/syscontrol/node4.html. Accessed: April 2008.
Feedback (http://www.fbk.com). Industrial process control. http://www.fbk.com/process control/. Accessed: April 2008.
NASA Tech Briefs (http://www.nasatech.com). Motion control technology. http://www.nasatech.com/motion/techbriefs.html. Accessed: April 2008. http://www.nasatech.com/motion/. Accessed: April 2008.
Danaher Motion (http://www.danahermotion.com). Motion control handbook. http://www.danahermotion.com/education training.html. Accessed: April 2008.
Stephen J. O'Neil. Motion control handbook. com.itmr.waw.servlet.pdf from Micro Mo Electronics, Inc. http://www.micromo.com. Accessed: May 2007. pp. 1 36.
Lee Stephens. Achieving positioning accuracy goal. http://www.nasatech.com/motion/features/feat 1206.html. Accessed: April 2008.
National Instruments (http://www.ni.com). NI developer zone. http://zone.ni.com/dzhp/app/main. Accessed: April 2008.
GlobalSpec (www.GlobalSpec.com). About manufacturing cells and systems. http://industrial automation.globalspec.com/LearnMore/Industrial Automation Services/Manufacturing Cells Systems? Accessed: May 2008.
RobotWorx (http://www.robots.com). Industrial robots and automation. http://www.robots.com/Industrial Robots and Robotic Automation Systems.mht. Accessed: May 2008.
K. Thramboulidis, C. Tranoris. An architecture for the development of function block oriented engineering support system. IEEE CIRA. Canada 2001.
AdvanTech (http://www.advantech.de). Industrial automation. http://www.advantech.de/eAutomation/. Accessed: May 2008.
Altera (http://www.altera.com). Industrial automation. http://www.altera.com/Automation.mht. Accessed: May 2008.

CHAPTER

Sensors and actuators

3

Industrial controllers, or computers in control systems monitor the operating states and working conditions of the perceive equipment. Through sensors, the controllers and computers vital information about states and conditions, which allows adjustments to be made far more quickly and accurately than mechanical systems can do them by themselves. Sensors convert measurements of temperature, pressure, density, strength, position, or other quantities into either digital or analog electric signals.

Industrial controllers or computers in control systems use data received from sensors to control or drive different equipment or systems by the use of actuators. The actuators are electromechanical devices which can generate electrical, magnetic, pneumatic, hydraulic, optical or other forces to drive shafts, or movements of the controlled equipment.

In many industrial control systems, those sensors and actuators constitute a network with connections of various types of Fieldbus. Industrial control systems can be very complex, and are usually structured into several hierarchical levels. The networks that consist of sensor and actuator (sensor-actuator networks) are specified as field level. Figure 3.1 shows a definition of the hierarchical levels of industrial control systems. According to this definition, an industrial control system can have the following levels: the management level, which provides human-system interfaces for production and business management; the control level, in which the controllers, robots and computers implement the control functions; the field level, which consists of sensor-actuator networks; and the equipment level, which includes the various pieces of equipment used on the factory floor.

The sensor-actuator networks in this field level generally include sensors, actuators, transducers and valves. This chapter describes the sensors and actuators. The next chapter describes transducers and valves.

3.1 INDUSTRIAL OPTICAL SENSORS

Optical sensors are devices that are used to measure temperature, pressure, strain, vibration, acoustics, rotation, acceleration, position, and water vapor content. The detector of an optical sensor measures changes in characteristics of light, such as intensity, polarization, time of flight, frequency (color), modal cross-talk or phase.

Optical sensors can be categorized as point sensors, integrated sensors, multiplexed sensors, and distributed sensors. This section specifies the three that are typically applied in industrial control systems; color sensors, light section sensors and scan sensors.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10003-8

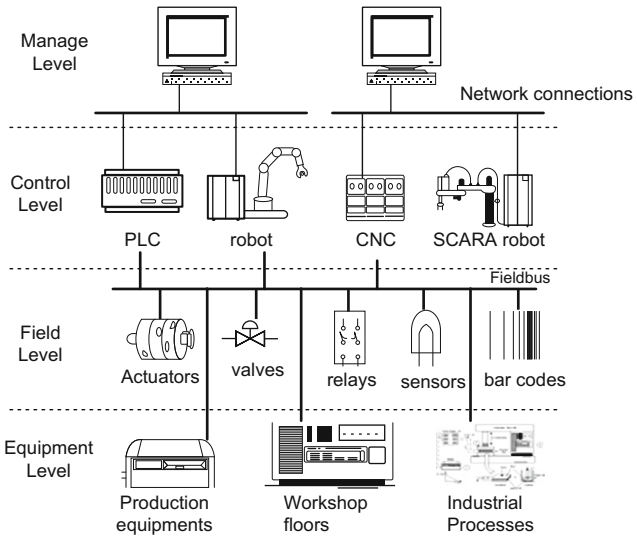


FIGURE 3.1

A definition of the hierarchical levels of industrial control systems.

3.1.1 Color sensors

Color sensors that can operate in real time under various environmental conditions benefit many applications, including quality control, chemical sensing, food production, medical diagnostics, energy conservation, and monitoring of hazardous waste. Analogous applications can also be found in other fields of the economy; for example, in the electric industry for recognition and assignment of colored cords, and for the automatic testing of mounted LED (light-emitting diode) arrays or matrices, in the textile industry to check coloring processes, or in the building materials industry to control compounding processes.

Such sensors are generally advisable wherever color structures, color processes, color nuances, or colored body rims must be recognized in homogeneously continuous processes over a long period, and where color has an influence on the process control or quality protection.

(1) Operating principle

Color detection occurs in color sensors according to the three-field (red, green, and blue) procedure. Color sensors cast light on the objects to be tested, then calculate the chromaticity coordinates from the reflected or transmitted radiation, and compare them with previously stored reference tristimulus (red, green, and blue) values. If the tristimulus values are within the set tolerance range, a switching output is activated. Color sensors can detect the color of opaque objects through reflection (incident light), and of transparent materials via transmitted light, if a reflector is mounted opposite the sensor.

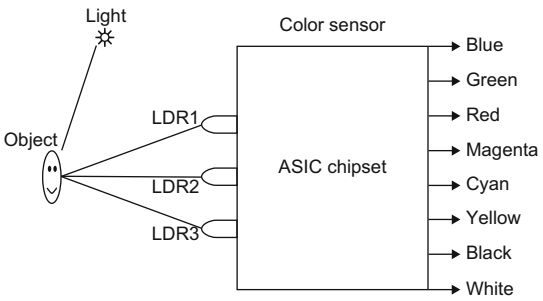


FIGURE 3.2

Operating principle of a proposed color sensor.

In Figure 3.2, the color sensor shown can sense eight colors: red, green, and blue (primary colors of light); magenta, yellow, and cyan (secondary colors); and black and white. The ASIC chipset of the color sensor is based on the fundamentals of optics and digital electronics. The object whose color is to be detected is placed in front of the system. Light reflected from the object will fall on the three convex lenses that are fixed in front of the three LDRs (light dependent resistors). The convex lenses will cause the incident light rays to converge. Red, green, and blue glass filters are fixed in front of LDR1, LDR2, and LDR3, respectively. When reflected light rays from the object fall on the gadget, the filter glass plates determine which of these three LDRs will be triggered. When a primary color falls on the system, the lens corresponding to that primary color will allow the light to pass through; but the other two glass plates will not allow any transmission. Thus, only one LDR will be triggered and the gate output corresponding to that LDR will indicate which color it is. Similarly, when a secondary color light ray falls on the system, the two lenses corresponding to the two primary colors that form the secondary color will allow that light to pass through while the remaining one will not allow any light ray to pass through it. As a result two of these three LDRs are triggered and the gate output corresponding to these indicates which color it is. All three LDRs are triggered for white light and none will be triggered for black, respectively.

(2) Basic types

Color sensors can be divided into three-field color sensors and structured color sensors according to how they work.

(a) Three-field color sensors

This sensor works based on the tristimulus (standard spectral) value function, and identifies colors with extremely high precision, 10,000 times faster than the human eye. It provides a compact and dynamic technical solution for general color detection and color measurement. It is capable of detecting, analyzing, and measuring minute differences in color, for example, as part of LED testing, calibration of monitors, or where mobile equipment is employed for color measurement. Based on the techniques used, there are two kinds of three-field color sensors:

(i) The three-element color sensor. This sensor includes special filters so that their output currents are proportional to the function of standard XYZ tristimulus values. The resulting absolute XYZ standard

spectral values can thus be used for further conversion into a selectable color space. This allows for a sufficiently broad range of accuracies in color detection from "eye accurate" to "true color", that is, standard-compliant colorimetry to match the various application environments.
(ii) The integral color sensor. This kind of sensor accommodates integrative features including (1) detection of color changes; (2) recognition of color labels; (3) sorting colored objects; (4) checking of color-sensitive production processes; and (5) control of the product appearance.

(b) Structured color sensors

Structured color sensors are used for the simultaneous recording of color and geometric information, including (1) determination of color edges or structures and (2) checking of industrial mixing and separation processes. There are two kinds of structured color sensors available:
(i) The row color sensor. This has been developed for detecting and controlling color codes and color sequences in the continuous measurement of moving objects. These color sensors are designed as PIN-photo-diode arrays. The photo diodes are arranged in the form of honeycombs for each of three rhombi. The diode lines consist of two honeycomb lines displaced a half-line relative to each other. As a result, it is possible to implement a high resolution of photo diode arrays in the most compact format. In this case, the detail to be controlled is determined by the choice of focusing optics.
(ii) The hexagonal color sensor. This kind of sensor generates information for the subsequent electronics about the three-field chrominance signal (intensity of the three-receiving segments covered with the spectral filters), as well as about the structure and the position.

(3) Application guide

In industrial control, color sensors are selected by means of the application-oriented principle, referring to the following technical factors; (1) operating voltage, (2) rated voltage, (3) output voltage, (4) residual ripple maximum, (5) no-load current, (6) spectral sensitivity, (7) size of measuring dot minimum, (8) limiting frequency, (9) color temperature, (10) light spot minimum, (11) permitted ambient temperature, (12) enclosure rating, (13) control interface type.

3.1.2 Light section sensors

Light section methods have been utilized as a three-dimensional measurement technique for many years, as non-contact geometry and contour control. The light section sensor is primarily used for automating industrial production processes, and testing procedures in which system-relevant positioning parameters are generated from profile information. Two-dimensional height information can be collected by means of laser light, the light section method, and a high resolution of receiver array. Height profiles can be monitored, the filling level detected, magazines counted, or the presence of objects checked.

(1) Operating principle

A light section is obtained using the laser light section method. This is a three-dimensional procedure to measure object profiles in a one-sectional plane. The principle of laser triangulation (see Figure 3.3) requires a camera positioned orthogonal to the object's surface area, to measure the lateral displacement, or deformation, of a laser line projected at an angle θ onto the object's surface (see

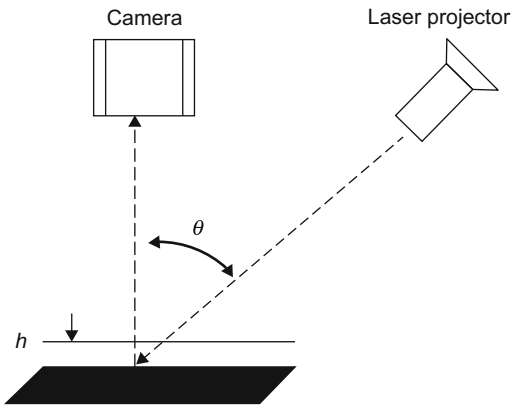


FIGURE 3.3

Laser triangulation (optical scheme), where h is elevation measurement range and θ is the angle between the plane of the laser line and the axis of the camera. A high resolution can be obtained by increasing h and decreasing θ and vice versa.

Figure 3.4). The elevation profile of interest is calculated from the deviation of the laser line from the zero position.

A light section sensor consists of a camera and a laser projector, also called a laser line generator. The measurement principle of a light section sensor is based on active triangulation (Figure 3.3). The simplest method is to scan a scene by a laser beam and detect the location of the reflected beam. A laser

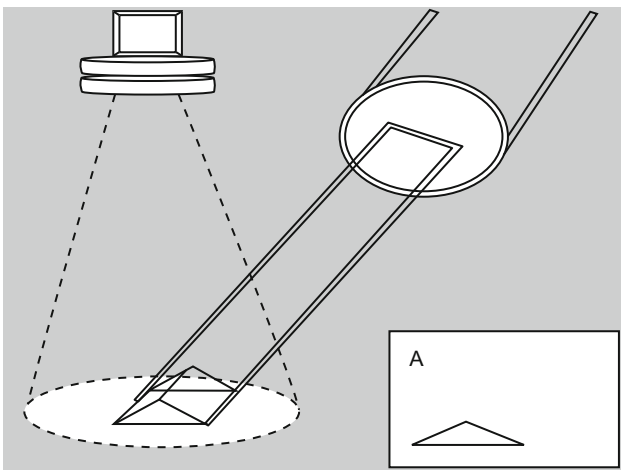


FIGURE 3.4

Laser light sectioning is the two-dimensional extension of the laser triangulation. By projecting the expanded laser line an elevation profile of the object under test is obtained. Inset A: image recorded by the area camera. The displacement of the laser line indicates the object elevation at the point of incidence.

beam can be spread by passing it through a cylindrical lens, which results in a plane of light. The profile of this light can be measured in the camera image, thus giving triangulation along one profile. In order to generate dense range images, one has to project several light planes (Figure 3.4). This can be achieved either by moving the projecting device, or by projecting many strips at once. In the latter case the strips have to be encoded; this is referred to as the coded-light approach. The simplest encoding is achieved by assigning different brightnesses to every projection direction, for example, by projecting a linear intensity ramp.

Measuring range and resolution are determined by the triangulation angle between the plane of the laser line and the optical axis of the camera (see Figure 3.3). The more obtuse this angle, the larger is the observed lateral displacement of the line. The measured resolution is increased, but the measured elevation range is reduced. Criteria related to an object's surface characteristics, camera aperture or depth of focus, and the width of the laser line can reduce the achievable resolution.

(2) Application guide

In applications, there are some technical details of the laser light section method that require particular attention.

(a) Object surface characteristics

One requirement for the utilization of the laser light section method is an at least partially diffuse reflecting surface. An ideal mirror would not reflect any laser radiation into the camera lens, and the camera cannot view the actual reflecting position of the laser light on the object surface. With a completely diffuse reflecting surface, the angular distribution of the reflected radiation is independent of the angle of incidence of the incoming radiation as it hits the object under test (Figure 3.4).

Real surfaces usually provide a mixture of diffuse and reflecting behavior. The diffuse reflected radiation is not distributed isotropically, which means that the more obtuse the incoming light, the smaller the amount of radiation is reflected in an orthogonal direction to the object's surface. Using the laser light section method, the reflection characteristics of the object's surface (depending on the submitted laser power and sensitivity of the camera) limit the achievable angle of triangulation θ (Figure 3.3).

(b) Depth of focus of the camera and lens

To ensure broadly constant signal amplitude at the sensor, the depth of focus of the camera lens, as well as the depth of focus of the laser line generator have to cover the complete elevation range of measurement.

By imaging the object under test onto the camera sensor, the depth of focus of the imaging lens increases proportionally to the aperture number k, the pixel distance y, following a quadratic relationship with the imaging factor @ (= field of view/sensor size).

The depth of focus 2z is calculated by:

$$2z = 2yk @ (1 + @)$$

In the range $\pm z$ around the optimum object distance, no reduction in sharpness of the image is evident.

Example:

Pixel distance y = 0.010 mm

Aperture number k = 8

Imaging factor @ = 3

$2z = 2 \times 0.010 \times 8 \times 3 \times (1 + 3) = 1.92$ mm

With fixed image geometry a diminishing aperture of the lens increases its depth of focus. A larger aperture number, k, cuts the signal amplitude by a factor of 2 with each aperture step; it decreases the optical resolution of the lens and increases the negative influence of speckle effect.

(c) Depth of focus of a laser line

The laser line is focused at a fixed working distance. If actual working distances diverge from the given setting, the laser line widens, and the power density of the radiation decreases.

The region around the nominal working distance, where line width does not increase by more than a given factor, is the depth of focus of a laser line. There are two types of laser line generators: laser micro line generators and laser macro line generators.

Laser micro line generators create thin laser lines with a Gaussian intensity profile that is orthogonal to the direction of propogation laser line. The depth of focus of a laser line at wavelength L and of width B is given by the so-called Rayleigh range:

$$2Z_R : 2Z_R = (\pi B^2)/(2L), \text{ where } \pi = 3.1415926$$

Laser macro line generators create laser lines with higher depth of focus. At the same working distance macro laser lines are wider than micro laser lines. Within the two design types, the respective line width is proportional to the working distance. Due to the theoretical connection between line width and depth of focus, the minimum line width of the laser line is limited by the application, due to the required depth of focus.

(d) Basic setback: laser speckle

Laser speckling is an interference phenomenon originating from the coherence of laser radiation; for example, the laser radiation reflected by a rough-textured surface.

Laser speckle disturbs the edge sharpness and homogeneity of laser lines. Orthogonal to the laser line, the center of intensity is displaced stochastically. The granularity of the speckle depends on the setting of the lens aperture viewing the object.

At low apertures the speckles have a high spatial frequency; with a large k number the speckles are rather rough and particularly disturbing. Since a diffuse, and thus optically rough-textured surface is essential for using a laser light section, laser speckling cannot be entirely avoided.

Reduction is possible by (1) utilizing laser beam sources with decreased coherence length, (2) moving object and sensor relative to each other, possibly using a necessary or existing movement of the sensor or the object (e.g., the profile measurement of railroad tracks while the train is running), (3) choosing large lens apertures (small aperture numbers), as long as the requirements of depth of focus can tolerate this.

(e) Dome illuminator for diffuse illumination

This application requires control of the object outline and surface at the same time as the three-dimensional profile measurement. For this purpose, the object under test is illuminated homogeneously and diffusely by a dome illuminator. An LED ring lamp generates light that scattered by a diffuse reflecting cupola to the object of interest. In the center of the dome an opening for the camera is located, so there is no radiation falling onto the object from this direction.

Shadow and glint are largely avoided. Because the circumstances correspond approximately to the illumination on a cloudy day, this kind of illumination is also called "cloudy day illumination".

(f) Optical engineering

The design of the system configuration is of great importance for a laser light section application high requirements, it implies the choice and dedicated optical design of such components as camera, lens, and laser line generator. Optimum picture recording within the given physical boundary conditions can be accomplished by consideration of the laws of optics. Elaborate picture preprocessing algorithms are avoided.

Measuring objects with largely diffuse reflecting surfaces by reducing or allows the use of requirements in resolution, cameras and laser line generators from an electronic mail order catalogue system testing (or for projects such as school practical, etc.).

These simple laser line generators mostly use a glass rod lens to produce a Gaussian intensity profile along the laser line (as mentioned in the operating principle section). As precision requirements increase, laser lines with largely constant intensity distribution and line width have to be utilized.

3.1.3 Scan sensors

Scan sensors, also called image sensors or vision sensors, are built for industrial applications. Common applications for these sensors in industrial control include alignment or guidance, assembly quality, bar or matrix code reading, biotechnology or medical, color mark or color recognition, container or product counting, edge detection, electronics or semiconductor inspection, electronics rework, flaw detection, food and beverage, gauging, scanning and dimensioning, ID detection or verification, materials analysis, noncontact profilometry, optical character recognition, parcel or baggage sorting, pattern recognition, pharmaceutical packaging, presence or absence, production and quality control, seal integrity, security and biometrics, tool and die monitoring, and website inspection.

(1) Operating principle

A scan or vision or image sensor can be thought of as an electronic input device that converts analog information from a document such as a map, a photograph, or an overlay, into an electronic image in a digital format that can be used by the computer. Scanning is the main operation of a scan, vision or image sensor, which automatically captures document features, text, and symbols as individual cells, or pixels, and so produces an electronic image.

While scanning, a bright white light strikes the image and is reflected onto the photosensitive surface of the sensor. Each pixel transfers a gray value (values given to the different shades of black in the image ranging from 0 (black) to 255 (white), that is, 256 values, to the chipset (software)). The software interprets the value in terms of 0 (black) or 1 (white) as a percentage, thereby forming a monochrome image of the scanned item. As the sensor moves forward, it scans the image in tiny strips and the sensor stores the information in a sequential fashion. The software running the scanner pieces together the information from the sensor into a digital image. This type of scanning is known as one-pass scanning.

Scanning a color image is slightly different, as the sensor has to scan the same image for three different colors; red, green, and blue (RGB). Nowadays most of the color sensors perfom one-pass scanning for all three colors in one go, by using color filters. In principle, a color sensor works in the

same way as a monochrome sensor, except that each color is constructed by mixing red, green, and blue as shown in Figure 3.5. A 24-bit RGB sensor represents each pixel by 24 bits of information. Usually, a sensor using these three colors (in full 24 RGB mode) can create up to 16.8 million colors.

Full width, single-line contact sensor array scanning is a new technology that has emerged, in which the document to be scanned passes under a line of chips which captures the image. In this technology, a scanned line could be considered as the cartography of the luminosity of points on the line observed by the sensor. This new technology enables the scanner to operate at previously unattainable speeds.

(2) CCD image sensors

A charge-coupled device (CCD) gets its name from the way the charges on its pixels are read after an exposure. After the exposure the charges on the first row are transferred to a place on the sensor called the read-out register. From there, the signals are fed to an amplifier and then on to an analog-to-digital converter. Once the row has been read, its charges on the read-out register row are deleted, the next row enters, and all of the rows above march down one row. The charges on each row are "coupled" to those on the row above so when the first moves down, the next also moves down to fill its old space. In this way, each row can be read one at a time. Figure 3.6 is a diagram of the CCD scanning process.

(3) CMOS image sensors

A complementary metal oxide semiconductor (CMOS) typically has an electronic rolling shutter design. In a CMOS sensor the data are not passed from bucket to bucket. Instead, each bucket can be read independently to the output. This has enabled designers to build an electronic rolling slit shutter. This shutter is typically implemented by causing a reset to an entire row and then, some time later, reading the row out. The readout speed limits the speed of the wave that passes over the sensor from top

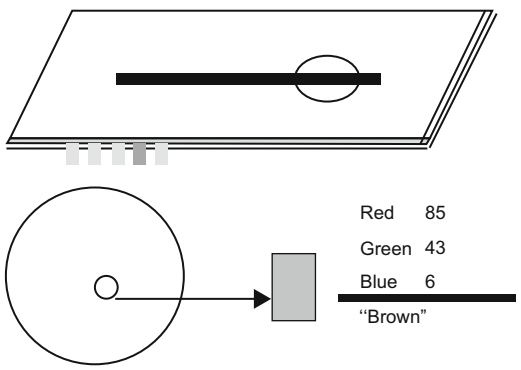


FIGURE 3.5

The scan sensor operation of the scanning a color image. In this figure, a pixel of red = 85, green = 43, and blue = 6 is being scanned, which is identified as brown.

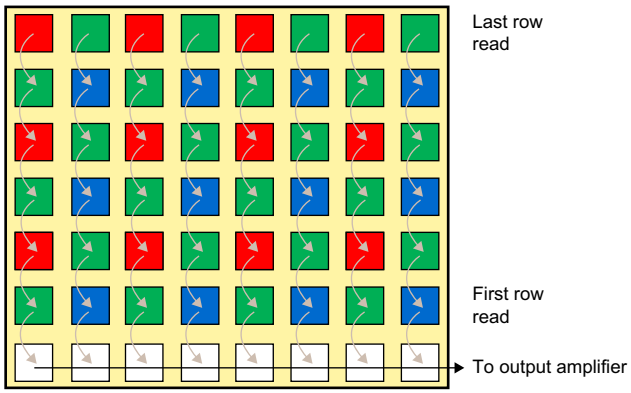


FIGURE 3.6

The CCD image sensor shifts one whole row at a time into the readout register. The readout register then shifts one pixel at a time to the output amplifier.

to bottom. If the readout wave is preceded by a similar wave of resets, then uniform exposure time for all rows is achieved (albeit not at the same time). With this type of electronic rolling shutter there is no need for a mechanical shutter except in certain cases. These advantages allow CMOS image sensors to be used in some of the highest specification industrial control devices or finest cameras. Figure 3.7 gives a typical architecture of industrial control devices with a CMOS image sensor.

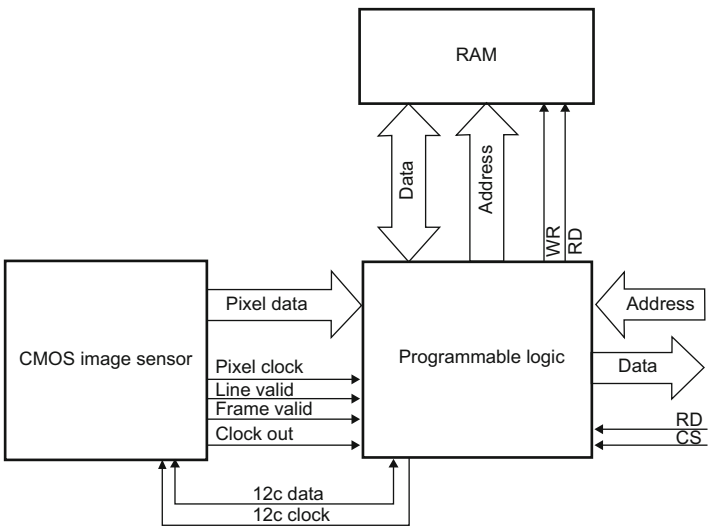


FIGURE 3.7

Block diagram of an industrial control device with a CMOS image sensor.

(2) Basic types

All scan (or image or vision) sensors can be monochrome or color sensors. Monochrome sensing sensors present the image in black and white or as a grayscales. Color sensing sensors can sense the spectrum range using varying combinations of discrete colors. One common technique is sensing the red, green, and blue components (RGB) and combining them to create a wide spectrum of colors. Multiple chip colors are available on some scan (image or vision) sensors. In a widely used method, the colors are captured in multiple chips, each of them being dedicated to capturing part of the color image, such as one color, and the results are combined to generate the full color image. They typically employ color separation devices such as beam-splitters, rather than having integral filters on the sensors.

The imaging technology used in scan or image or vision sensors includes CCD, CMOS, tube, and film.

(a) CCD image sensors (charge-coupled device) are electronic devices that are capable of transforming a light pattern (image) into an electric charge pattern (an electronic image). The CCD consists of several individual elements that have the capability of collecting, storing, and transporting electrical charges from one element to another. This together with the photosensitive properties of silicon is used to design image sensors. Each photosensitive element will then represent a picture element (pixel). Using semiconductor technologies and design rules, structures are made that form lines or matrices of pixels. One or more output amplifiers at the edge of the chip collect the signals from the CCD. After exposing the sensor to a light pattern, a series of pulses that transfer the charge of one pixel after another to the output amplifier are applied, line after line. The output amplifier converts the charge into a voltage. External electronics then transform this output signal into a form that is suitable for monitors or frame grabbers.

CCD image sensors have extremely low noise levels and can act either as color sensors or monochrome sensors.

Choices for array type include linear array, frame transfer area array, full-frame area array, and interline transfer area array. Digital imaging optical format is a measure of the size of the imaging area. It is used to determine what size of lens is necessary, and refers to the length of the diagonal of the imaging area. Optical format choices include 1/7, 1/6, 1/5, 1/4, 1/3, 1/2, 2/3, 3/4, and 1 inch. The number of pixels and pixel size are also important. Horizontal pixels refer to the number of pixels in each row of the image sensor. Vertical pixels refer to the number of pixels in a column of the image sensor. The greater the number of pixels, the higher the resolution of the image.

Important image sensor performance specifications to consider when considering CCD image sensors include spectral response, data rate, quantum efficiency, dynamic range, and number of outputs. The spectral response is the spectral range (wavelength range) for which the detector is designed. The data rate is the speed of the data transfer process, normally expressed in megahertz.

Quantum efficiency is the ratio of photon-generated electrons that the pixel captures to the photons incident on the pixel area. This value is wavelength-dependent, so the given value for quantum efficiency is generally for the peak sensitivity wavelength of the CCD. Dynamic range is the logarithmic ratio of well depth to the readout noise in decibels: the higher the number, the better.

Common features for CCD image sensors include antiblooming and cooling. Some arrays for CCD image sensors offer an optional antiblooming gate designed to bleed off overflow from a saturated pixel. Without this feature, a bright spot which has saturated the pixels will cause a vertical streak in

the resultant image. Some arrays are cooled for lower noise and higher sensitivity, so to consider an important environmental parameter operating temperature.

(b) CMOS image sensors operate at lower voltages than CCD image sensors, reducing power consumption for portable applications. In addition, CMOS image sensors are generally of much simpler design: often just a crystal and decoupling circuits. For this reason, they are easier to design with, generally smaller, and require less support circuitry.

Each CMOS active pixel sensor cell has its own buffer amplifier that can be addressed and read individually. A commonly used design has four transistors and a photo-sensing element. The cell has a transfer gate separating the photo sensor from a capacitive floating diffusion, a reset gate between the floating diffusion and power supply, a source-follower transistor to buffer the floating diffusion from readout-line capacitance, and a row-select gate to connect the cell to the readout line. All pixels on a column connect to a common sense amplifier. In addition to, sensing in color or in monochrome, CMOS sensors are divided into two categories of output; analog or digital. Analog sensors output their encoded signal in a video format that can be fed directly to standard video equipment. Digital CMOS image sensors provide digital output, typically via a 4/8- or 16-bit bus. The digital signal is direct, not requiring transfer or conversion via a video capture card.

CMOS image sensors can offer many advantages over CCD image sensors. Just some of these are (1) no blooming, (2) low power consumption, ideal for battery-operated devices, (3) direct digital output (incorporates ADC and associated circuitry), (4) small size and little support circuitry, and (5) simple to design.

(c) The tube camera is an electronic device in which the image is formed on a fluorescent screen. It is then read by an electron beam in a raster scan pattern and converted to a voltage proportional to the image light intensity.

(d) Film technology exposes the image onto photosensitive film, which is then developed to play or store. The shutter, a manual door that admits light to the film, typically controls exposure.

3.2 INDUSTRIAL PHYSICAL SENSORS

Physical sensors measure the variations in physical properties of the samples, including temperature, moisture, density, thermal diffusivity, stiffness, thickness, bending stiffness, shear rigidity, elastic modulus, presence of irregularities (cracks, voids, and edges), defects, and strength etc. In theory, all measurable physical properties can be used in physical sensing applications, thus generating many possible physical sensors.

This section explains two kinds of physical sensors: temperature and distance sensors, which are important in industrial control systems.

3.2.1 Temperature sensors

Big differences exist between different temperature sensor or temperature measurement device types. Using one perspective, they can be simply classified into two groups, contact and noncontact.

Contact temperature sensors measure their own temperature. The temperature of the object to which the sensor is in contact (is infered by assuming) that the two are in thermal equilibrium. Noncontact temperature sensors infer the temperature of an object by measuring the radiant power that is assumed to be emitted (some may be reflected rather than emitted). Table 3.1 lists some contact and noncontact temperature sensors.

The following paragraphs in this subsection introduce a contact temperature sensor: the bimetallic sensor.

(1) Operating principles of bimetallic sensors

Bimetallic sensors are electromechanical thermal sensors, or limiters, that are used for automatic temperature monitoring in industrial control. They limit the temperature of machines or devices by opening up the power load or electric circuit in the case of overheating, or by shutting off a ventilator or activating an alarm in the case of overcooling.

Bimetal sensors can also serve as time-delay devices. The usual technique is to pass current through a heater coil that eventually (after 10 seconds or so) warms the bimetal elements enough to actuate. This is the method employed on some controllers such as the cold-start fuel valves found on automobile engines.

A bimetallic sensor essentially consists of two metal strips fixed together. If the two metals have different coefficients of thermal expansion, then as the temperature of the switch changes, one strip will expand more than the other, causing the device to bend out of plane. This mechanical bending can then be used to actuate an electromechanical switch or be part of an electrical circuit itself, so that contact of the bimetallic device with an electrode causes a circuit to be made. Figure 3.8 is a diagrammatic representation of the typical operation of temperature switches. There are two directional processes given in this diagram, which cause the contacts to change from open to closed and from closed to open, respectively:

(a) Event starts; the time is zero, the temperature of the sensor is T_1, and the contacts are open. As the environment temperature increases, the sensor is abruptly heated and reaches the temperature T_2 at some point, causing the contacts to close.

(b) When the temperature of the sensor is T_2 and the contacts are closed, if the environment temperature keeps decreasing, the sensor is abruptly cooled, which takes the temperature to T_1 at some point, causing the contacts to open again.

(2) Basic types of bimetallic sensors

Bimetallic sensors basically fall into two broad categories: creep action devices with slow make and slow break switching action; and snap action devices with quick make and quick break switching action.

Creep action devices are excellent either in temperature-control applications or as a high-limit controls. They have a narrow temperature differential between opening and closing, and generally have more rapid cycling characteristics than snap action devices.

Snap action devices are most often used for temperature-limiting applications, as their fairly wide differential between opening and closing temperature provides slower cycling characteristics.

Table 3.1 Temperature Sensors

Types	Main Subtypes	Working Mechanism	Industrial Applications
Contact temperature sensors	Thermocouples	Thermocouples are pairs of dissimilar metal wires or metal rods joined at one or two ends, which generate a net thermoelectric voltage between the open pair according to the size of the temperature difference between the ends, based on the Seebeck effect that occurs in electrical conductors.	Thermocouples are the easiest temperature sensors to make and are widely used in science and industry.
	Resistance temperature detectors	Resistance temperature detectors are wire wound and thin film devices that measure temperature because of the physical principle of the positive temperature coefficient of electrical resistance of metals.	Resistance temperature detectors are provided encapsulated in probes or other devices for temperature sensing and measurement with an external indicator, controller or transmitter.
	Thermometers	Thermometers measure the phase changes when their specified temperatures are exceeded, when they can change color or temperature crayons melt and become liquid. They are single-use devices although they may have multiple points on a single label.	Thermometers are found in forms such as temperature labels or temperature stickers having a central white or yellowish dot that turns black when the temperature value printed on the label is exceeded.
	Thermowells	Thermowells are normally made from drilled molybdenum rods with an internal sheath of high-purity alumina. The annular space between the alumina and metal had a very slow gas purge of nitrogen + hydrogen to prevent oxidation of the moly surface. Thermowells can be inserted into the bottom of an electrically heated glass melting furnace to measure the molten glass temperature.	Thermowells are applied for industrial temperature measurements. There are many variations of two basic kinds; low pressure and high pressure. They are used to provide isolation between a temperature sensor and the environment such as liquid, gas or slurry.

	Thermistors	A thermistor is a thermally sensitive resistor that exhibits a change in electrical resistance with a change in its temperature. The resistance is measured by passing a small, measured direct current through it and measuring the voltage drop produced.	Thermistors behave like temperature-sensitive electrical resistors. There are basically two types: negative temperature coefficient, used in temperature sensing and positive temperature coefficient, used in electric current control.
Noncontact temperature sensors	Radiation thermometers	Radiation thermometers measure temperature from the amount of thermal electromagnetic radiation received from a spot on the object of measurement.	Radiation thermometers enable automation and feedback control.They save lives and improve safety in fire-fighting, rescue, and detection of criminal activities.
	Infrared thermograph	Thermal imaging devices can collect and convert the thermal infrared radiation emitted (and also reflected) by objects into images that can be seen on a view screen or computer display.	Thermal imagers are widely used for temperature measurement in many non-destructive testing situations.
	Optical pyrometer	Optical pyrometers work on the basic principle of using the human eye to match the brightness of the hot object to the brightness of a calibrated lamp filament inside the instrument. The optical system contains filters that restrict the wavelength-sensitivity of the devices to a narrow wavelength band around 0.65 to 0.66 microns.	The optical pyrometer is a highly-developed and well accepted noncontact temperature measurement device

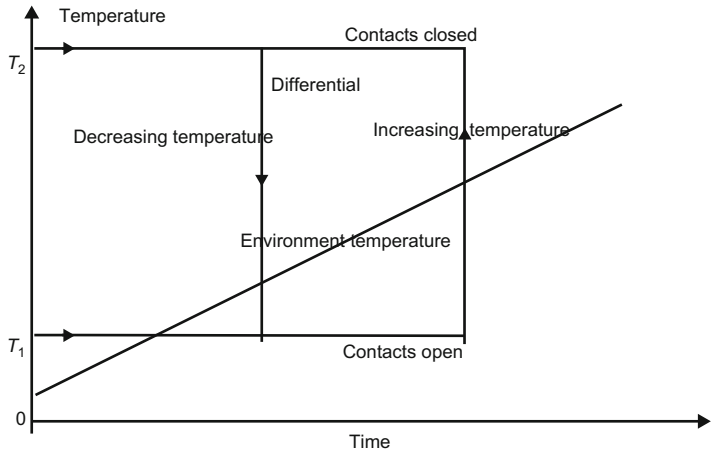


FIGURE 3.8

Typical operation of temperature sensors.

Although in its simplest form a bimetallic sensor can be constructed from two flat pieces of metal, in practical terms a whole range of shapes are used to provide maximum actuation or maximum force during thermal cycling. As shown in Figure 3.9, the bimetallic elements can be of three configurations in a bimetallic sensor:

(a) In Figure 3.9(a), two metals make up the bimetallic strip (hence the name). In this diagram, the black metal would be chosen to expand faster than the white metal if the device were being used in an oven, so that as the temperature rises the black metal expands faster. This causes the strip to bend downward, breaking contact so that current is cut off. In a refrigerator you would use the opposite set-up, so that as the temperature rises the white metal expands faster than the black metal. This causes the strip to bend upward, making contact so that current can flow. By adjusting the size of the gap between the strip and the contact, you can control the temperature.

(b) Another configuration uses a bimetallic element as a plunger, or pushrod, to force contacts open or closed. Here the bimetal does not twist or deflect, but instead is designed to lengthen or travel as a means of actuation, as illustrated by Figure 3.9(b). Bimetallic sensors can be designed to switch at a wide range of temperatures. The simplest devices have a single set-point temperature determined by the geometry of the bimetal and switch packaging. Examples include switches found in consumer products. More sophisticated devices for industrial use may incorporate calibration mechanisms for adjusting temperature sensitivity or switch-response times. These mechanisms typically use the separation between contacts as a means of changing the operating parameters.

(c) Bimetal elements can also be disk-shaped, as in Figure 3.9(c). These types often incorporate a dimple as a means of producing a snap action (not shown in this figure). Disk configurations tend to handle shock and vibration better than cantilevered bimetallic sensors.

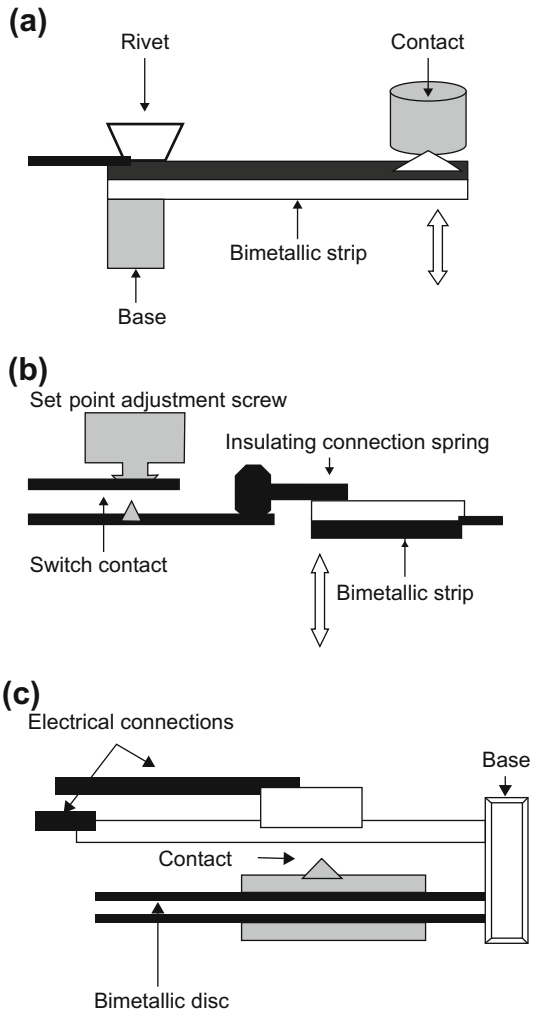


FIGURE 3.9

The operating principle for bimetallic sensors: (a) basic bimetallic sensor, (b) adjustable set-point sensor, and (c) bimetallic disc sensor.

3.2.2 Distance sensors

There are several kinds of distance sensors that work based on different physical mechanisms, including optical, acoustic, capacitive, inductive and photoelectric physics. These distance sensors can be classified into two groups; passive and active. A passive sensor does not emit, but just measures the change of a physical field around the sensor. In contrast, an active sensor not only emits but also measures the change of a physical field around the sensor.

Table 3.2 lists the main types of distance sensors. However, this subsection will focus on an active distance sensor; the ultrasonic distance sensor.

(1) Operating principles of ultrasonic sensors

Ultrasonic distance sensors measure the distance to, or presence of target objects by sending a pulsed ultrasound wave at the object and then measuring the time for the sound echo to return. Knowing the speed of sound, the sensor can determine the distance of the target object.

As illustrated in Figure 3.10, the ultrasonic distance sensor regularly emits a barely audible click. It does this by briefly supplying a high voltage either to a piezoelectric crystal, or to the magnetic fields of ferromagnetic materials. In the first case, the crystal bends and sends out a sound wave. A timer within the sensor keeps track of exactly how long it takes the sound wave to bounce off a target and return. This delay is then converted into a voltage that corresponds to the distance from the sensed object.

In the second case, the physical response of a ferromagnetic material in a magnetic field is due to the presence of magnetic moments. Interaction of an external magnetic field with the domains causes a magnetostrictive effect. Controlling the ordering of the domains through alloy selection, thermal annealing, cold working, and magnetic field strength can optimize this effect. The magnetostrictive effects are produced by the use of magnetostrictive bars to control high-frequency oscillators and to produce ultrasonic waves in gases, liquids, and solids.

Applying converters based on the reversible piezoelectric effect makes one-head systems possible, where the converter serves both as transmitter and as receiver. The transceivers work by transmitting a short-burst ultrasonic packet. An internal clock starts simultaneously, measuring propagation time. The clock stops when the sound packet is received back at the sensor. The time elapsed between transmitting the packet and receiving the echo forms the basis for calculating distance. Complete control of the process is realized by an integrated microcontroller, which allows excellent output linearity.

(2) Basic types of ultrasonic sensors

The ultrasonic distance sensor can be operated in two different modes. The first mode, referred to as continuous (or analog) mode, involves the sensor continuously sending out sound waves at a rate determined by the manufacturer. The second mode, called clock (or digital) mode, involves the sensor sending out signals at a rate determined by the user. This rate can be several signals per second with the use of a timing device, or it can be triggered intermittently by an event such as the press of a button.

The major benefit of ultrasonic distance sensors is their ability to measure difficult targets; solids, liquids, powders, and even transparent and highly reflective materials that would cause problems for optical sensors. In addition, analog output ultrasonic sensors offer comparatively long ranges, in many cases > 3 m. They can also be very small some tubular models are only 12 mm in diameter, and 15 mm $\times$ 20 mm $\times$ 49 mm square-bodied versions are available for limited-space applications.

Ultrasonic devices do have some limitations. Foam, or other attenuating surfaces may absorb most of the sound, significantly decreasing the measuring range. Extremely rough surfaces may diffuse the sound excessively, decreasing range and resolution. However, an optimal resolution is usually guaranteed up to a surface roughness of 0.2 mm. Ultrasonic sensors emit a wide sonic cone, limiting their usefulness for small target measurement and increasing the chance of receiving feedback from interfering objects. Some ultrasonic devices offer a sonic cone angle as narrow as 6 degrees, permitting

Table 3.2 Distance Sensors

Types	Working Mechanism	Industrial Applications
Optical distance sensors	Optical distance sensors detect the decrease or change in transmission of light emitted from a laser or diode. Optical sensors utilize various spectral regions. Infrared sensors avoid interference from ambient light. Red light sensors offer a visible beam that can be sensitive to the colour of the target. Laser sensors measure the interruption of emitted laser.	Optical sensors can be used for highly accurate distance measurements. Interruption of an optical beam can be used to confirm the presence of such as pipette tips or filters, without disrupting the normal flow of system operations.
Acoustic distance sensors	Ultrasonic sensors are the most important acoustic distance sensors. Ultrasonic sensors are noncontact devices which use the sonar principal in air, sending out an ultrasonic chirp, then switching to the receive mode to detect a return echo from the surface of the target. With the speed of sound in air (or other gas) as a given, distance to the target can be calculated.	Turbulence, foam, vapors, and changes in the concentration of the process material also affect the ultrasonic sensor's response. This technology can be used for measuring: wind speed and direction (anemometer), fullness of a tank, and speed through air or water.
Inductive distance sensors	Inductive sensors are noncontact devices that set up a radio frequency field with an oscillator and a coil. The presence of an object alters this field and the sensor is able to detect this alteration. This field is emitted at the sensing face of the sensor. If a metallic object (switching trigger) nears the sensing face, eddy currents are generated. An inductive sensor comprises an oscillating circuit, a signal evaluator, and a switching amplifier. The coil of this oscillating circuit generates a high-frequency electromagnetic alternating field.	Inductive analog sensors are ideal for applications involving relatively short metal target travel. They provide precision, noncontact position measurement by varying the electrical output in proportion to the position of a metal target within working range.

(*Continued*)

Table 3.2 Distance Sensors *Continued*

Types	Working Mechanism	Industrial Applications
Capacitive distance sensors	Capacitive distance sensors are noncontact sensors that work by measuring changes in an electrical property called capacitance. Capacitance describes how two conductive objects with a space between them respond to a voltage difference applied to them. When a voltage is applied to the conductors, an electric field is created between them causing positive and negative charges to collect on each object. If the polarity of the voltage is reversed, the charges will also reverse.	The capacitance is directly proportional to the surface area of the objects and the dielectric constant of the material between them and inversely proportional to the distance between them. In typical capacitive sensing applications, the probe or sensor is one of the conductive objects; the target object is the other. Therefore, any change in capacitance is a result of a change in the distance between the probe and the target.
Photoelectric distance sensors	Photoelectric distance sensors emit irradiating light to the target, receive the reflected light from the target, and can make measurement of the target position. Photoelectric sensors are made up of a light source, a receiver, a signal converter, and an amplifier. The phototransistor analyzes incoming light, verifies that it is from the light source, and appropriately triggers an output. Through beam photoelectric sensors are configured with the emitter and detector opposite the path of the target and sense presence when the beam is broken. Retroreflective photoelectric sensors are configured with the emitter and detector and rely on a reflector to bounce the beam back across the path of the target.	Photoelectric distance sensors are specially used for detecting small objects at a long distance. Photoelectric sensors offer many advantages when compared to other technologies. Sensing ranges for photoelectric sensors far surpass the inductive, capacitive, magnetic, and ultrasonic technologies.

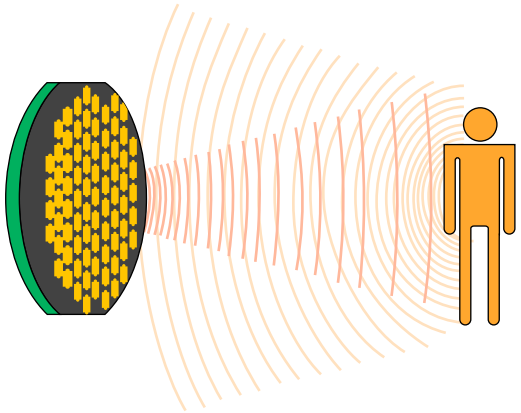


FIGURE 3.10

Operating principle of ultrasonic distance sensors.

detection of very small objects, and sensing of targets through narrow spaces such as bottlenecks, pipes, and ampoules.

3.3 INDUSTRIAL MEASUREMENT SENSORS

Sensors for measurement in industrial applications provide a comprehensive and practical introduction to the increasingly important topic of sensor technology, and also serves as an extensive guide to sensors and their applications. Table 3.3 gives the main types of industrial measurement sensors. Due to space restrictions, this section briefly explains only two measurement sensors; force and load sensors.

3.3.1 Force sensors

The most common dynamic force and acceleration detector is the piezoelectric sensor. A piezoelectric sensor produces a voltage when it is squeezed by a force. This voltage is proportional to the force applied. The fundamental difference between such devices and static force detection devices is that the electrical signal generated by the crystal decays rapidly after the application of force. This makes piezoelectric sensors unsuitable for the detection of static force.

Depending on the application requirements, dynamic force can be measured as either compressive, tensile, or torque force. Applications may include the measurement of spring or sliding friction forces, chain tensions, clutch release forces, or peel strengths of laminates, labels, and pull tabs.

The high-impedance electrical signal generated by the piezoelectric crystal is converted (by an amplifier) to a low-impedance signal suitable for such an instrument as a digital storage oscilloscope. Digital storage of the signal is required in order to allow analysis of the signal before it decays. The low-impedance voltage mode (LIVM) force sensor, discussed below, is a typical example of a piezoelectric force sensor.

Table 3.3 Measurement Sensors

Types	Working Mechanism	Industrial Applications
Position measurement sensors	A position measurement sensor is a physical or mechanical device that enables either an absolute or a relative (displacement), and either a linear or an angular position measurement. Position measurement can be performed in a contact or noncontact manner. Position sensors can be based on optical, acoustic, photoelectric, electric, magnetic, or electromagnetic mechanisms.	Position measurement sensors are applied for position or displacement, motor replacements, proximity detection, valve positioning, shaft travel, automotive steering, robotics, and throttle position systems for industries such as automotive, aviation etc.
Level measurement sensors	Level sensors detect the level of substances including liquids, slurries, granular materials, and powders. All such substances flow to become essentially level in their containers (or other physical boundaries) due to gravity. Capacitance, conductance, hydrostatic tank gauging, radar, and ultrasonic are the leading technologies for level measurement.	The level measurement can be either continuous or point values. Continuous level sensors measure the level within a specified range and determine the exact amount of substance in a certain place, while point-level sensors only indicate whether the substance is above or below the sensing point.
Speed measurement sensors	Speed measurement sensors dynamically read the motion speeds of objects. Wheel speed sensors are the main type of speed sensors, which have sender devices used for reading the speed of a vehicle wheel rotation. It usually consists of a toothed ring and pickup. The physical mechanisms for sensing the speed can be optical, magnetic, and Doppler, etc., working in a contact or noncontact manner.	Speed sensors can be applied in an electronically controlled suspension system, a power steering system, an anti-lock brake system, a traction control system, an electronic stability program, etc. and are critical components of vehicle anti-lock braking systems.
Dimension measurement sensors	The dimension sensor, or profile sensor, consists of an optical camera system with the functionality of a photoelectric switch. In operation, a laser beam generates a long line of light over a target object at operating distances. It permits distance-independent profile detection and the measurement field fluctuates.	Profile sensors detect and scan objects that cross the sensing area beam by beam, and use these signals to detect the shape and size of the object, making this sensor ideal for a variety of applications, such as sorting, inspection, and motion control.

Pressure measurement sensors	A pressure sensor measures pressure, typically of gases or liquids. Pressure is an expression of the force required to stop a fluid from expanding in terms of force per unit area. A pressure sensor generates a signal related to the pressure imposed. Typically, such a signal is electrical; however, optical, visual, and auditory signals are also used.	Pressure sensors are widely used for control applications to indirectly measure other variables such as fluid/gas flow, speed, water level, and altitude.
Force measurement sensors	Force sensors are required for an accurate determination of pulling and/or pressing forces. The force sensor outputs an electrical signal which corresponds to the force measurement for further evaluating or controlling processes. Many force sensors use piezoelectric measuring elements to convert a mechanical stress into an electrical resistance change.	Force sensors are commonly used in automotive vehicles and a variety of subsystems such as brake, suspension, transmission, and speed control and safety systems. For example, a force sensor is installed in air bag systems fitted to cars to provide safety for a passenger in the event of a collision.
Load measurement sensors	Load sensors are electromechanical transducers that translate force or weight into voltage. The change in voltage produces, in the read-out instrumentation, a repeatable deflection or indication that can be calibrated directly in terms of the load applied to the load transducer.	A load is a vital part of many industrial processes. In automatically controlled systems, it is desirable to monitor the forces and moments being generated at the work site between the motive, or drive power, element, and the driven element of the operating unit.

(1) Construction and operating principles

Figure 3.11(a) shows a typical LIVM force sensor with radial connector. Figure 3.11(b) shows an axial connector sensor.

Two quartz discs are preloaded together between a lower base and an upper platen by means of an elastic preload screw (or stud) as seen in Figure 3.11(a) and (b). Preloading is necessary to ensure that the crystals are held in close contact, for best linearity and to allow a tension range for the instruments. In the radial connector style (Figure 3.11(a)), both platen and base are tapped to receive threaded members such as mounting studs, impact caps or machine elements. Platen and base are welded to an outer housing which encloses and protects the crystals from the extemal environment. A thin, steel web connects the platen to the outer housing, allowing the quartz element structure to flex unimpeded by the housing structure. The integral amplifier is located in the radially mounted connector housing.

Construction of the axial connector type (Figure 3.11(b)) is similar to the radial connector type, except that the lower base contains a threaded, integral, mounting stud, which also serves as the amplifier housing, and supports the electrical connector. This design allows the electrical connection to exit axially, which is especially useful where radial space is limited.

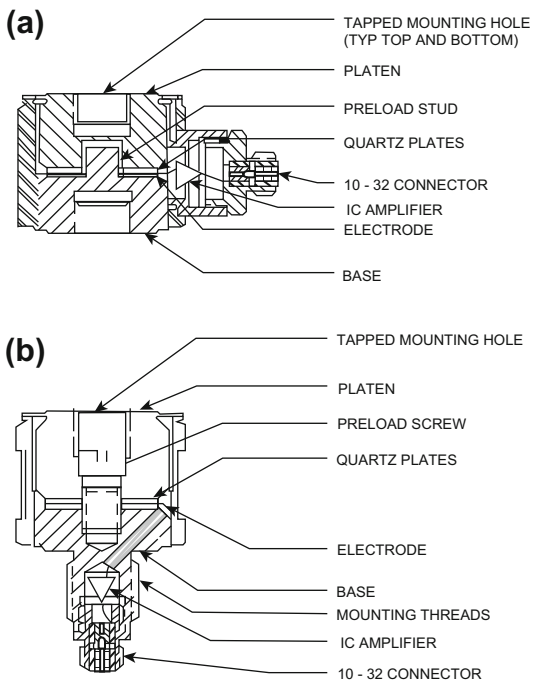


FIGURE 3.11

Low-impedance voltage mode (LIVM) force sensors.

(Courtesy of DYTRAN Instrument, Inc. 2009.)

When the crystals are stressed by an external compressive force, an analogous positive polarity voltage is generated. This voltage is collected by the electrode and connected to the input of a metal oxide silicon field effect transistor unity gain source follower amplifier located within the amplifier housing. The amplifier serves to lower the output impedance of the signal by 10 orders of magnitude, so that it can be displayed on readout instruments such as oscilloscopes, meters or recorders. When the sensor is put under tensile loads (pulled), some of the preload is released, causing the crystals to generate a negative output signal. Maximum tensile loading is limited by the ultimate strength of the internal preload screw and is usually much less than the compression range.

(2) Application guides

Because they are almost as rigid as a comparably proportioned piece of solid steel, piezoelectric force sensors may be inserted directly into machines as part of their structure. By virtue of this high rigidity, these sensors have very high natural frequencies with fast rise time capabilities, making them ideal for measuring very quick transient forces, such as those generated by metal-to-metal impacts and high-frequency vibrations.

Although LIVM force sensors are designed to measure dynamic forces, the discharge time constants of most units are long enough to allow static calibration, by which we mean the use of calibrated weights or ring dynamometers. An important rule of thumb for this type of calibration is that the first 10% of the discharge time constant curve is relatively linear with respect to time. What this means is that the output signal will decay 1% in 1% of the discharge time constant, and so on up to about 10 seconds. This tells us that, in order to make a reading that is accurate to 1% (other measurement errors not considered), we must take our reading within 1% of the discharge time constant (in seconds), after application of the calibration force.

The most convenient way to do this is by use of a digital storage oscilloscope and a DC (direct current) coupled current source power unit. The DC coupled unit is essential because the AC (alternating current) coupling of conventional power units would make the overall system coupling time-constant too short to perform an accurate calibration in most cases.

3.3.2 Load sensors

The most common technology used by load sensors uses the principle of strain gauges.

In a photoelectric strain gauge, a beam of light is passed through a variable slit, actuated by the extensometer, and directed to a photoelectric cell. As the gap opening changes, the amount of light reaching the cell varies, causing a matching intensity change in the current generated by the cell. Semiconductor or piezoelectric strain gauges are constructed of ferroelectric materials, such crystalline quartz. In such materials a change in the electronic charge across the faces of the crystal occurs when it is mechanically stressed. The piezoresistive effect is defined as the change in resistance of a material due to given applied stress, and this term is commonly used in connection with semiconducting materials. Optical strain gauge types include photoelastic, moiré interferometer, and holographic interferometer strain gauges. In a fiber-optic strain gauge, the sensor measures the strain by shifting the frequency of the light reflected down the fiber from the Bragg grating, which is embedded inside the fiber itself.

The gauge pattern refers cumulatively to the shape of the grid, the number and orientation of the grids in a multiple grid (rosette) gauge, the solder tab configuration, and various construction features

that are standard for a particular pattern. Arrangement types include uniaxial, dual linear, strip gauges, diaphragm, tee rosette, rectangular rosette, and delta rosette. Specialty applications for strain gauges include crack detection, crack propagation, extensometer, temperature measurement, residual stress, shear modulus gauge, and transducer gauge.

The three primary specifications when selecting strain gauges are operating temperature, the statistics of the strain (including gradient, magnitude, and time dependence), and the stability required by the application. The operating temperature range is the range of ambient temperature where the use of the strain gauge is permitted without permanent changes of the measurement properties. Other important parameters to consider include the active gauge length, the gauge factor, nominal resistance, and strain-sensitive material. The gauge length of a strain gauge is the active or strain-sensitive length of the grid. The end loops and solder tabs are considered insensitive to strain because of their relatively large cross-sectional area and low electrical resistance.

The strain sensitivity of a strain gauge is the ratio between the relative changes in resistance. The strain sensitivity is dimension less and is generally called the gauge factor. The resistance of a strain gauge is defined as the electrical resistance measured between the two metal ribbons or contact areas intended for the connection of measurement cables. The principal component that determines the operating characteristics of a strain gauge is the strain-sensitive material used in the foil grid.

3.4 INDUSTRIAL ACTUATORS

Industrial actuators are categorized by energy source. This section will discuss five actuators which use different types of energy; electric, magnetic, pneumatic, hydraulic and piezoelectric actuators.

3.4.1 Electric actuators

Electric actuators, which use the simplicity of electrical operation, provide the most reliable means of positioning a valve in a safe condition, including a fail-safe to closed or open, or a lock in position on power or system failure. However, electric actuators are not restricted to open or close applications; with the addition of one or more of the available options, the requirements of fully fledged control units can often be met. For example, with both weatherproof and flameproof models, the range simplifies process automation by providing true electronic control from process variable to valve and supplies a totally electric system for all environments. The unit can be supplied with the appropriate electronic controls to match any process control system requirement. More and more often, electric actuators provide a superior solution, especially when high accuracy, high duty cycle, excellent reliability, long life expectancy, and low maintenance are supplied by extra switches, speed controllers, potentiometers, position transmitters, positioners, and local control stations. These options may be added to factory-built units, or supplied in kit form. When supplied as kits, all parts are included together with an easy to follow installation sheet.

(1) Operating principle

The architecture for an electric actuator is given in Figure 3.12, and consists basically of gears, a motor, and switches. In these components, the motor plays a key role. In most applications, the motor is the primary torque-generating component. Motors are available for a variety of supply voltages, including standard single-phase alternating current, three-phase and DC voltages. In some applications, three-phase current for the asynchronous electric actuator is generated by means of the power circuit module, regardless of the power supply (one or three phase). Frequency converters and microcontrollers allow different speeds and precise tripping torques to be set (no over-torque). When an electric actuator is running, the phase angle is checked and automatically adjusted, so that the rotation is always correct. To prevent heat damage due to excessive current draw in a stalled condition, or due to overwork, electric actuator motors usually include a thermal overload sensor or switch embedded in the winding of the stator. The sensor or switch is installed in series with the power source, and opens the circuit when the motor is overheated, and then closes the circuit once it has cooled to a safe operating temperature.

Electric actuators rely on a gear train (a series of interconnected gears) to enhance the motor torque and to regulate the output speed of the actuator. Some gear styles are inherently self-locking. This is particularly important in the automation of butterfly valves, or when an electric actuator is used in modulating control applications. In these situations, seat and disk contact, or fluid velocity, act upon the closure element of the valve and causes a reverse force that can reverse the motor and camshaft. This causes a re-energization of the motor through the limit switch when the cam position is changed. This undesirable cycling will continue to occur unless a motor brake is installed, and usually leads to an overheated motor. Spur gears are sometimes used in rotary electric actuators, but are not self-locking. They require the addition of an electromechanical motor brake for these applications.

One self-locking gear types is a worm and wheel, with some configurations of planetary gears. A basic worm gear system operates as follows: a motor applies a force through the primary worm gear to

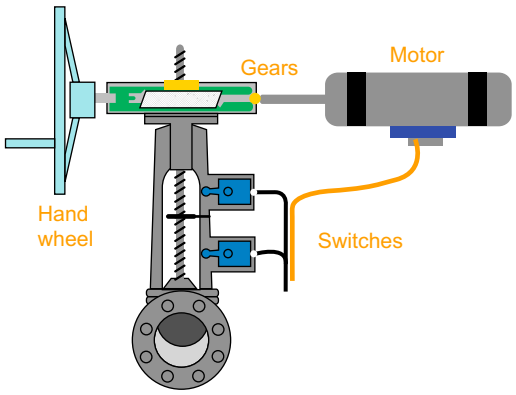


FIGURE 3.12

Basic components of an electric actuator.

the worm wheel. This, in turn, rotates the secondary worm gear which applies a force to the larger radius of the secondary worm wheel to increase the torque.

(2) Basic types

Electric actuators are divided into two different types; rotary and linear. Rotary electric actuators rotate from open to closed using butterfly, ball, and plug valves. With the use of rotary electric actuators, the electromagnetic power from the motor causes the components to rotate, allowing for numerous stops during each stroke. Either a circular shaft or a table can be used as the rotational element. When selecting an electric rotary actuator, the actuator torque and range of motion should be considered. The actuator torque refers to the power that causes the rotation, while the full range of motion can be either nominal, quarter-turn, or multiturn. Linear electric actuators, in contrast, open and close using pinch, globe, diaphragm, gate, or angle valves. They are often used when tight tolerances are required. These electric actuators use an acme screw assembly or motor-driven ball screw to supply linear motion. In linear electric actuators, the load is connected to the end of a screw that is belt or gear driven. Important factors to consider when selecting linear electric actuators include the number of turns, actuating force, and the length of the valve stem stroke.

(a) Linear electric actuators provide linear motion via a motor-driven ball screw or screw assembly. The linear actuator's load is attached to the end of a screw, or rod, and is unsupported. The screw can be direct, belt, or gear driven. Important performance specifications to consider when considering for linear actuators include stroke, maximum rated load or force, maximum rated speed, continuous power, and system backlash. Stroke is the distance between fully extended and fully retracted rod positions. The maximum rated load or force is not the maximum static load. The maximum rated speed is the maximum actuator linear speed, typically rated at low or no load. Continuous power is sustainable power; it does not include short-term peak power ratings. Backlash is position error due to direction change. Motor choices include DC (direct current), DC servo, DC brushless, DC brushless servo, AC (alternating current), AC servo, and stepper. Input power can be specified for DC, AC, or stepper motors.

Drive screw specifications for linear actuators include drive screw type and screw lead. Features include self-locking, limit switches, motor encoder feedback, and linear position feedback. Screw choices include acme screws and ball screws. Acme screws will typically hold loads without power, but are usually less efficient than ball screws. They also typically have a shorter life but are more robust to shock loads. If backlash is a concern, it is usually better to select a ball screw. Ball screws exhibit lower friction and therefore higher efficiency than lead screws. Screw lead is the distance the rod advances with one revolution of the screw.

(b) Rotary electric actuators provide incremental rotational movement of the output shaft. In its most simple form, a rotary actuator consists of a motor with a speed reducer. These AC and DC motors can be fabricated to the exact voltage, frequency, power, and performance specified. The speed reducer is matched with the ratio to the speed, torque, and acceleration required. Life, duty cycle, limit load, and accuracy are considerations that further define the selection of the speed reducer. Hardened, precision spur gears are supported by antifriction bearings as a standard practice in these speed reducers. Compound gear reduction is accomplished in compact, multiple load path configurations, as well as in planetary forms. The specifications for rotary actuator include angular rotation, torque, and speed, as well as control signals and feedback signals, and the environment temperature.

Rotary actuators can incorporate a variety of auxiliary components such as brakes, clutches, antibacklash gears, and/or special seals. Redundant schemes involving velocity or torque summing of two or more motors can also be employed. Today the linear motion in actuators is converted to a rotary one in many applications. By delivering the rotary motion directly, some fittings can be saved in the bed. This enables the bed manufacturer to build in a rotary actuator far more elegantly than a linear actuator. The result is a purer design, because the actuator is not seen as a product hanging under the bed, but as a part of the bed.

Rotary electric actuators are used for modulating valves, which are divided based on the range from multiturn to quarter turn. Electrically powered multiturn actuators are one of the most commonly used and dependable configurations of actuators. A single or three-phase electric motor drives a combination of spurs and/or level gears, which in turn drive a stem nut. The stem nut engages the stem of the valve to open or close it, frequently via an acme threaded shaft. Electric multiturn actuators are capable of operating very large valves quickly. To protect the valve, the limit switch turns off the motor at the ends of travel. The torque-sensing mechanism of the actuator switches off the electric motor when a safe torque level is exceeded. Position-indicating switches are utilized to indicate the open and closed position of the valve. Typically a declutching mechanism and hand wheel are also included so that the valve can be operated manually should a power failure occur.

Electric quarter-turn actuators are very similar to electric multiturn actuators. The main difference is that the final drive element is usually in one quadrant that puts out a 90° motion. The latest generation of quarter-turn actuators incorporates many of the features found in most sophisticated multiturn actuators, for example, a nonintrusive, infrared, human machine interface for set-up, diagnostics, etc. Quarter-turn electric actuators are compact and can be used on smaller valves. They are typically rated to around 1500 foot pounds. An added advantage of smaller quarter-turn actuators is that, because of their lower power requirements, they can be fitted with an emergency power source, such as a battery, to provide fail-safe operation.

Thrust actuators can be fitted to valves which require a linear movement. Thrust actuators transform the torque of a multi-turn actuator into an axial thrust by means of an integrated thrust unit. The required (switch-off) actuating force (thrust and traction) can be adjusted continuously and reproducibly. Linear actuators are mainly used to operate globe valves. Thrust units, fitted to the output drive of a multiturn actuator, consist mainly of a threaded spindle, a metric screw bolt to join the valve shaft, and a housing to protect the spindle against environmental influences. The described version is used for direct mounting of the actuator to the valve. However, fork joint thrust actuators (indirect mounting) can also operate butterfly valves or dampers, when direct mounting of a part-turn actuator is not possible or efficient. The thrust units of the thrust actuators for modulating duty also comply with the high demands of the modulating duty. Also, for these thrust units, high-quality materials and accurate tolerances ensure perfect function over many years of operation. The thrust units are operated by modulating actuators.

3.4.2 Magnetic actuators

Magnetic control systems predominantly consist of magnetic field sensors, magnetic switches, and actuators that measure magnetic fields and or magnetic flux by evaluating a potential, current, or resistance change due to the field strength and direction. They are used to study the magnetic field or flux around the Earth, permanent magnets, coils, and electrical devices in displacement measurement,

railway inspection systems, and as a linear potentiometer replacement, etc. Magnetic field sensors, switches and actuators can measure and control these properties without physical contact and have become the eyes of many industrial and navigation control systems.

Magnetic field sensors indirectly measure properties such as direction, position, rotation, angle, and current, by detecting a magnetic field and its changes. The first application of a permanent magnet was a third-century B.C. Chinese compass, which is a direction sensor. Compared to other direct methods such as optical or mechanical sensors, most magnetic field sensors require some signal processing to produce the property of interest. However, they provide reliable data without physical contact even in adverse conditions such as dirt, vibration, moisture, hazardous gas and oil, etc. At present, there are two kinds of magnetic switches; magnetic reed switches and magnetic level switches. Magnetic switches are suitable for applications requiring a switched output for proximity, linear limit detection, logging or counting, or actuation purposes.

A unique aspect of using magnetic sensors and switches is that measuring a magnetic field is usually not the primary intention. Another parameter is usually the objective, such as wheel speed, presence of a magnetic ink, vehicle detection, or heading determination, etc. These parameters cannot be measured directly but can be extracted from changes or disturbances in magnetic fields.

The most widely used magnetic sensors and switches are Hall effect sensor switches, magneto-resistive (MR) series of sensors switches, magnetic reed switches and magnetic level switches. The Hall effect is used in a device that converts the energy stored in a magnetic field to an electrical signal by means of the development of a voltage between the two edges of a current-carrying conductor whose faces are perpendicular to a magnetic field.

(1) Hall effect sensors and switches

The Hall effect is a conduction phenomenon which is different for different charge carriers. In most common electrical applications, conventional current is used partly because it makes no difference whether positive or negative charge is considered to be moving. But the Hall effect voltage has a different polarity for positive and negative charge carriers, and it has been used to study the details of conduction in semiconductors and other materials which show a combination of negative and positive charge carriers.

The Hall effect can be used to measure the average drift velocity of the charge carriers by mechanically moving the Hall probe at different speeds until the Hall voltage disappears, showing that the charge carriers are now not moving with respect to the magnetic field. Other types of investigations of carrier behavior are studied in the quantum Hall effect. An alternative application of the Hall effect is that it can be used to measure magnetic fields with a Hall probe.

As shown in Figure 3.13, if an electric current flows through a conductor in a magnetic field, the magnetic field exerts a transverse force on the moving charge carriers, which tends to push them to one side of the conductor. This is most evident in a thin, flat conductor as illustrated. A build-up of charge at the sides of the conductors will balance this magnetic influence, producing a measurable voltage between the two sides of the conductor. The presence of this measurable transverse voltage is called the Hall effect, after E. H. Hall, who discovered it in 1879.

Note that the direction of the current I in the diagram is that of conventional current, so that the motion of electrons is in the opposite direction. This further confuses all the right-hand rule manipulations needed to obtain the direction of the forces.

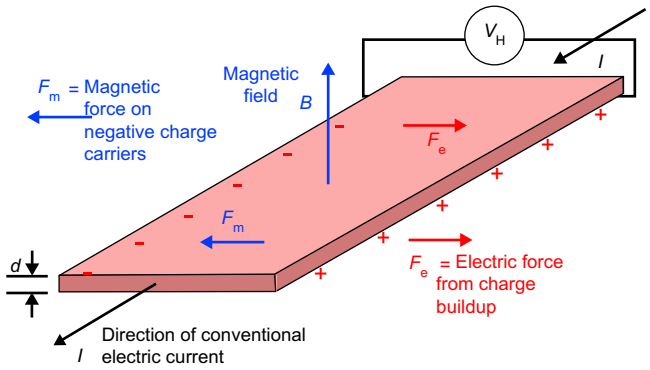


FIGURE 3.13

The Hall effect.

As displayed in Figure 3.13, the Hall voltage V_H is given by $V_H = IB/(ned)$, where I is the induced electric current, B is the strength of the magnetic field, n is the density of mobile charges, e is the electron charge, and d is the thickness of the film.

In the Hall effect sensor, the Hall effect element, with its entire evaluation circuitry, is integrated on a single silicon chip. The Hall effect plate with the current terminals and the taps for the Hall effect voltage are arranged on the surface of the crystal. This sensor element detects the components of the magnetic flux perpendicular to the surface of the chip, and emits a proportional electrical signal which is processed in the evaluation circuits which are integrated on the sensor chip. The functional principle of a Hall effect sensor is, as shown in Figure 3.14, that the output voltage of the sensor and the switching state, respectively, depend on the magnetic flux density through the Hall effect plate.

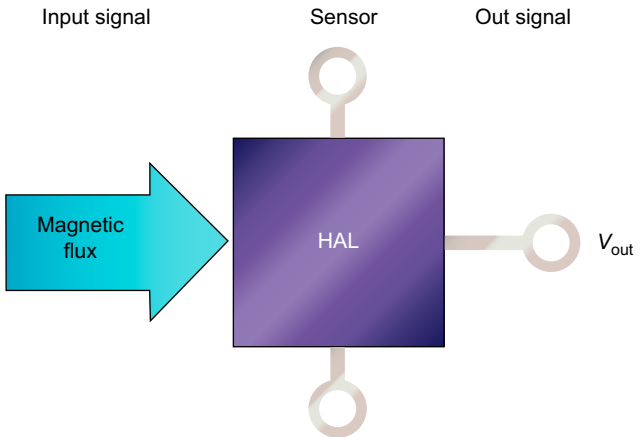


FIGURE 3.14

The functional principle of a Hall effect sensor.

(2) Magnetoresistive sensors and switches

Magnetoresistance is the ability of some conductive materials to gain or lose some of their electrical resistance when placed inside a magnetic field. The resistivity of some such materials is greatly affected when the material is subjected to a magnetic field. The magnitude of this effect is known as magnetoresistance and can be expressed by the equation:

$$\mathrm{MR} = (\mathrm{r(H)} - \mathrm{r(0)})/\mathrm{r(0)}.$$

where MR is the magnetoresistance, r(0) is the resistivity at zero magnetic fields, and r(H) is the resistivity in an applied magnetic field.

The magnetoresistance of conventional materials is quite small, but some materials with large magnetoresistance have now been synthesized. Depending on the magnitude of the effect produced, it is called either giant magnetoresistance (GMR) or colossal magnetoresistance (CMR).

Magnetoresistive sensor or switch elements are magnetically controllable resistors. The effect whereby the resistance of a thin, anisotropic, ferromagnetic layer changes through a magnetic field is utilized in these elements. The determining factor for the specific resistance is the angle formed by the internal direction of magnetization (M) and the direction of the current flow (I). Resistance is largest if the current flow (I) and the direction of magnetization run parallel. The resistance in the base material is smallest at an angle of 90° between the current flow (I) and the direction of magnetization (M).

Highly conductive material is then joined to it at an angle of less than, 45°. The current passing the sensor element takes the shortest distance between these two ranges. This means that it flows at a preferred direction of 45° against the longitudinal axis of the sensor element. Without an external field, the resistance of the element is then in the middle of its range. An external magnetic field with field strength (H) influences the internal direction of magnetization, which causes the resistance to change in proportion to its influence. Figure 3.15 gives an example using a Perm alloy (NiFe) film to illustrate that the resistance of a material depends upon the angle between the internal direction of magnetization M and the direction of the current flow (I).

The actual sensor element is often designed with four magnetic field sensitive resistors, that are interconnected to form a measuring bridge (Figure 3.16). The measuring bridge is energized and supplies a bridge voltage. A magnetic field, which influences the bridge branches by different degrees results in a voltage difference between the bridge branches which is then amplified and evaluated.

The sensor detects the movement of ferromagnetic structures (e.g., in gearwheels) caused by the changes in the magnetic flow. The sensor element is biased with a permanent magnet. A tooth or a gap moving past the sensor influences the magnetic field by different amounts. This causes changes in the magnetic field which depend on resistance values in a magnetoresistive sensor. The changes in the magnetic field can therefore be converted into an electric variable, and can also be conditioned accordingly. The output signal from the sensor is a square-wave voltage which reflects the changes in the magnetic field.

The voltage changes that result from the magnetic field changes are amplified and supplied to a Schmitt trigger after conditioning. If the effective signal reaches an adequate level, the output stage is set accordingly.

The sensor is used for noncontact rotational speed detection on ferromagnetic sensing objects such as gearwheels. The distance between the sensed object and the surface of the active sensor is described

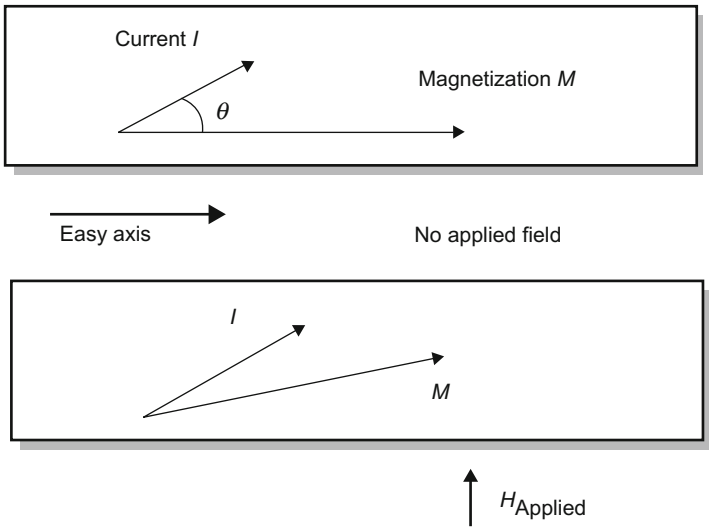


FIGURE 3.15

As an example, the Permalloy (NiFe) film has a magnetization vector, M, that is influenced by the applied magnetic field being measured. The resistance of the film changes as a function of the angle between the vector M and current I flowing through it. This change in resistance is known as the magnetoresistive effect.

as an air gap. The maximum possible air gap is dependent on the geometry of the object. The measurement principle dictates direction-dependent installation. The magnetoresistive sensor is sensitive to changes in the external magnetic field. For this reason the sensed objects should not have different degrees of magnetization.

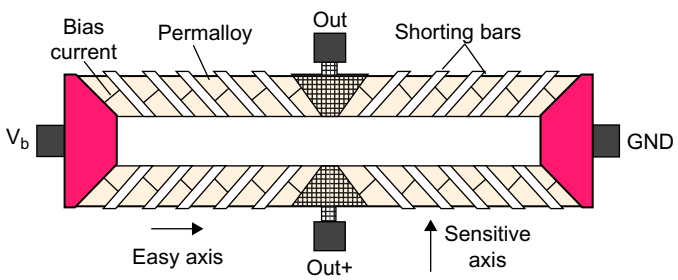


FIGURE 3.16

As an example, the magnetoresistive bridge is made up of four Permalloy parallel strips. A crosshatch pattern of metal is overlaid onto the strips to form shorting bars. The current then flows through the Permalloy, taking the shortest path, at a 45° angle from shorting bar to shorting bar. This establishes the bias angle between the magnetization vector M of the film and the current I flowing through it.

(3) Magnetic switches

Most magnetic switches work by using one of two mechanisms; magnetic reed switches and magnetic level switches.

Magnetic reed switches normally consist of two overlapping flat contacts, which are sealed into a glass tube that is filled with inert gas. When approached by a permanent magnet, the contact ends attract each other and make contact. When the magnet is removed, the contacts separate immediately (Figure 3.17).

Magnetic level switches use the time-proven principle of repelling magnetic forces. One permanent magnet forms part of a float assembly which rises and falls with changing liquid level. A second permanent magnet is positioned within the switch head so that the adjacent poles of the two magnets repel each other through a nonmagnetic diaphragm. A change in liquid level moves the float through its permissible range of travel, causing the float magnet to pivot and repel the switch magnet. The resulting snap action of the repelling magnets actuates the switch.

3.4.3 Pneumatic actuators

Pneumatics have had a variety of applications in control of industrial processes, from automotive and aircraft settings to modulate valves, to medical equipment such as dentistry drills to actuate torque movements. In contrast with other physical principles such as electrics or hydraulics, the operating torque of pneumatics makes possible a compact actuator that is economical to both install and operate. Pneumatic devices are also used where electric motors cannot be used for safety reasons, and where no water is supplied, such as in mining applications where rock drills are powered by air motors to preclude the need for electricity deep in the mine where explosive gases may be present. In many cases, it is easier to use a liquid or gas at high pressure rather than electricity to provide power for the actuator. Pneumatic actuators provide a very fast response but little power, whereas hydraulic systems can provide very great forces, but act more slowly. This is partly because gases are compressible and liquids are not. Pneumatic actuators (Figure 3.18) can offer the latest technology; a premium quality ball valve, a quality actuator designed to meet the torque requirements of the valve, and a mounting system which ensures alignment and rigidity.

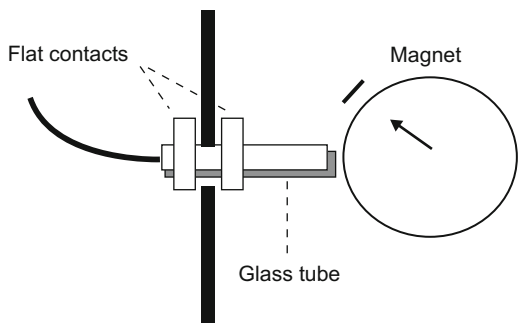


FIGURE 3.17

Operating principle of a simple magnetic reed switch.

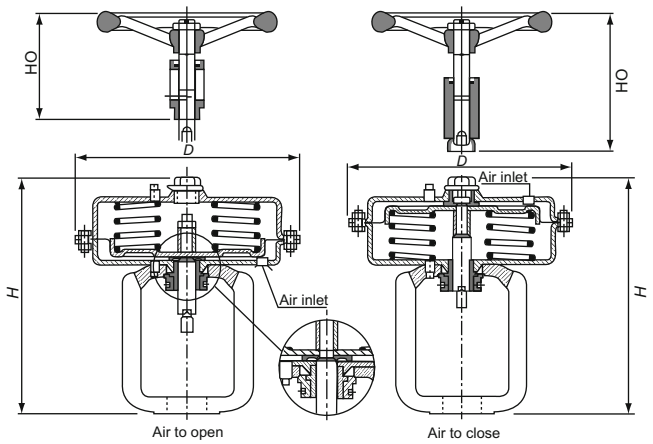


FIGURE 3.18

Components of a spring-type pneumatic actuator.

Industrial pneumatics may be contrasted with hydraulics, which uses uncompressible liquids such as oil, or water combined with soluble oil, instead of air. Air is compressible, and therefore it is considered to be a fluid. Pneumatic principles dictate that the pressure formed in compressible liquids can be harnessed to give a high potential of power. Harnessing this potential has lead to the development of a number of pneumatic-powered operations in new areas. Both pneumatics and hydraulics are applications of fluid power.

Both linear and rotary pneumatic actuators use pressurized air to drive or rotate mechanical components. The flow of pressurized air produces the shift or rotation of moving components via a stem and spring, rack and pinion, cams, direct air or fluid pressure on a chamber or rotary vanes, or other mechanical linkage. A valve actuator is a device mounted on a valve that, in response to a signal, automatically moves the valve to the desired position using an outside power source. Pneumatic valve actuators convert air pressure into motion.

(1) Linear pneumatic actuators

A linear pneumatic actuator operates a combination of force created by air and by the force of a spring. The actuator shifts the positions of a control valve, by transmitting its motion through the stem. A rubber diaphragm separates the actuator housing into two air chambers. The left or upper chamber receives a supply of air through an opening in the top of the housing. The right or bottom chamber contains a spring that forces the diaphragm against mechanical stops in the upper chamber. Finally, a local indicator is connected to the stem to indicate the position of the valve.

The position of the valve is controlled by varying supplied air pressure in the left or upper chamber, which results in a varying force acting on the top of the diaphragm. Initially, without supply air, the spring forces the diaphragm upward against the mechanical stops and holds the valve fully open. As supply air pressure is increased from zero, its force on top of the diaphragm begins to overcome the opposing force of the spring. The causes the diaphragm to move rightward or downward, and the control valve to close. With increasing supply air pressure, the diaphragm will continue to move

rightward or downward and compress the spring until the control valve is fully closed. Conversely, if supply air pressure is decreased, the spring will force the diaphragm leftward or upward and open the control valve. Additionally, if supply pressure is held constant at some value between zero and maximum, the valve will position at an intermediate point.

The valve can hence be positioned anywhere between fully open and fully closed in response to changes in the pressur of supplied air. A positioner is a device that regulates the supply air pressure to a pneumatic actuator. It does this by comparing the position demanded by the actuator with the control position of the valve. The requested position is transmitted by a pneumatic or electrical control signal from a controller to the positioner. The controller generates an output signal that represents the requested position. This signal is sent to the positioner. Externally, the positioner consists of an input connection for the control signal, a supply air input connection, a supply air output connection, a supply air vent connection, and a feedback linkage. Internally, it contains an intricate network of electrical transducers, air lines, valves, linkages, and necessary adjustments.

Other positioners may also provide controls for local valve positioning and gauges to indicate supply air pressure and controller pressure.

(2) Rotary pneumatic actuators

Pneumatic rotary actuators may have fixed or adjustable angular strokes, and can include such features as mechanical cushioning, closed-loop hydraulic dampening (by oil), and magnetic features for reading by a switch.

When the compressed air enters the actuator from the first tube nozzle, the air will push the double pistons toward both ends (cylinder end) for straight line movement. The gear on the piston causes the gear on the rotary shaft to rotate counterclockwise, and then the valve can open. This time, the air at both ends of the pneumatic actuator will drain through another tube nozzle.

Conversely, when the compressed air enters the actuator from the second tube nozzle, the gas will push the double pistons toward the middle for straight line movement. The rack on the piston drives the gear on the rotary shaft to rotate clockwise and then the valve can be closed. This time, the air at the middle of the pneumatic actuator will drain through the first tube nozzle. This is the standard driving principle. if required, a pneumatic actuator can be assembled into a driving principle that diffels from the standard type, which means opening the valve while rotating the rotary shaft clockwise and closing the valve while rotating the rotary shaft counterclockwise.

3.4.4 Hydraulic actuators

Pneumatic actuators are normally used to control processes that require a quick and accurate response, as they do not require a large amount of motive force. However, when a large amount of force is required to operate a valve, such as for main steam system valves, hydraulic actuators are normally used. A hydraulic actuator receives pressure energy and converts it to mechanical force and motion. Fluid power systems are manufactured by many organizations for a very wide range of applications, which often embody differing arrangements of components to provide the control functions required for the operation of diverse systems.

Table 3.4 gives the basic types of hydraulic actuators in three columns, (cylinder, valve, and motor), and the valve controlling source in three rows, (electro, servo, and piezoelectric). The motion type for each type of hydraulic actuator device is specified in each cell of the table, which indicates that

Table 3.4 Basic Types of Hydraulic Actuators

Types of Valve Controller	Hydraulic Cylinder Actuator	Hydraulic Valve Actuators	Hydraulic Motor Actuators
Electro	Linear	Linear rotary	Rotary
Servo	Linear	Linear rotary	Rotary
Piezoelectric	Linear	Linear rotary	Rotary

cylinders only have linear motion, valves can have both linear and rotary motions, and motors rotary motion only.

An actuator can be linear or rotary. A linear actuator gives force and motion outputs in a straight line. It is more commonly called a cylinder but is also referred to as a ram, reciprocating motor, or linear motor. A rotary actuator produces torque and rotating motion. It is more commonly called a hydraulic motor.

(1) Hydraulic cylinders and linear actuators

Hydraulic cylinders are actuation devices that use pressurized hydraulic fluid to produce linear motion and force. They are used in a variety of power transfer applications, and can be single or double action. A single action hydraulic cylinder is pressurized for motion in only one direction, whereas a double action hydraulic cylinder can move along the horizontal (x-axis) plane, the vertical (y-axis) plane, or along any other plane of motion.

Important operating specifications for hydraulic cylinders include the cylinder type, stroke, maximum operating pressure, bore diameter, and rod diameter. Stroke is the distance that the piston travels through the cylinder. Hydraulic cylinders can have a variety of stroke lengths, from fractions of an inch to many feet. The maximum operating pressure is the maximum working pressure the cylinder can sustain. The bore diameter refers to the diameter at the cylinder bore. The rod diameter refers to the diameter of the rod or piston used in the cylinder.

Choices for cylinder type include tie-rod, welded, and ram. A tie-rod cylinder is a hydraulic cylinder that uses one or more tie-rods to provide additional stability. Tie-rods are typically installed on the outside diameter of the cylinder housing. In many applications, the cylinder tie-rod bears the majority of the applied load. A welded cylinder is a smooth hydraulic cylinder that uses a heavy-duty welded cylinder housing to provide stability. A ram cylinder is a type of hydraulic cylinder that acts as a ram. A hydraulic ram is a device in which the cross-sectional area of the piston rod is more than one-half of the cross-sectional area of the moving components. Hydraulic rams are primarily used to push rather than pull, and are most commonly used in high-pressure applications.

(2) Hydraulic valves

Hydraulic valve actuators convert a fluid pressure supply into a motion. A valve actuator is a hydraulic actuator mounted on a valve that, in response to a signal, automatically moves the valve to the desired position using an outside power source. The actuators in hydraulic valves can be either linear, like cylinders, or rotary, like motors. The hydraulic actuator operates under servo-valve control; this provides regulated hydraulic fluid flow in a closed-loop system that has upper and lower cushions to

protect the actuator from the effects of high-speed and high-mass loads. Piston movement is monitored via a linear voltage displacement transducer (LVDT), which provides an output voltage proportional to the displacement of the movable core extension to the actuator.

The outside power sources used by hydraulic valves are normally electronic, servo, piezoelectric.

(3) Hydraulic motors and rotary actuators

Hydraulic motors are powered by pressurized hydraulic fluid and transfer rotational kinetic energy to mechanical devices. Hydraulic motors, when powered by a mechanical source, can rotate in the reverse direction, and act as a pump.

Hydraulic rotary actuators use pressurized fluid to rotate mechanical components. The flow of fluid produces the rotation of moving components via a rack and pinion, cams, direct fluid pressure on rotary vanes, or other mechanical linkage. Hydraulic rotary actuators and pneumatic rotary actuators may have fixed or adjustable angular strokes, and can include such features as mechanical cushioning, closed-loop hydraulic dampening (oil), and magnetic features for reading by a switch.

Motor type is the most important consideration when looking for hydraulic motors. The choices include axial piston, radial piston, internal gear, external gear, and vane. An axial piston motor uses an axially mounted piston to generate mechanical energy. High-pressure flow into the motor forces the piston to move in the chamber, generating output torque. A radial-piston hydraulic motor uses pistons mounted radially about a central axis to generate energy. An alternate-form radial-piston motor uses multiple interconnected pistons, usually in a star pattern, to generate energy. Oil supply enters the piston chambers, moving each individual piston and generating torque. Multiple pistons increase the displacement per revolution through the motor, increasing the output torque. An internal gear motor uses internal gears to produce mechanical energy. Pressurized fluid turns the internal gears, producing output torque. An external gear motor uses externally mounted gears to produce mechanical energy. Pressurized fluid forces the external gears to turn, producing output torque. A vane motor uses a vane to generate mechanical energy. Pressurized fluid strikes the blades in the vane, causing it to rotate and produce output torque.

Additional operating specifications to consider include operating torque, pressure, speed, temperature, power, maximum fluid flow, maximum fluid viscosity, displacement per revolution, and motor weight. The operating torque is the torque that the motor is capable of delivering, which depends directly on the pressure of the working fluid delivered to the motor. The operating pressure is the pressure of the working fluid delivered to the hydraulic motor. The fluid is pressurized by an outside source before it is delivered to the motor. Working pressure affects operating torque, speed, flow, and horsepower of the motor. The operating speed is the speed at which the hydraulic motors' moving parts rotate. Operating speed is expressed in terms of revolutions per minute or similar. The operating temperature is the fluid temperature range that the motor can accommodate. Minimum and maximum operating temperatures are dependent on the internal component materials of the motor, and can vary greatly between products. The power the motor is capable of delivering is dependent on the pressure and flow of the fluid through the motor. The maximum volumetric flow through the motor is expressed in terms of gallons per minute, or similar units. The maximum fluid viscosity the motor can accommodate is a measure of the fluid's resistance to shear, and is measured in centipoise (cP), a common metric unit of dynamic viscosity equal to 0.01 poise or 1 mP. The dynamic viscosity of water at 20°C is about 1 cP (the correct unit is cP, but cPs and cPo are sometimes used). The fluid volume displaced per

revolution of the motor is measured in cubic centimetres (cc) per revolution, or similar units. The weight of the motor is measured in pounds or similar units.

3.4.5 Piezoelectric actuators

Piezoelectric actuators represent an important new group of actuators for active control of mechanical systems. Although the magnitudes of piezoelectric voltages, movements, or forces are small, and often require amplification (e.g., a typical disk of piezoelectric ceramic will increase or decrease in thickness by only a small fraction of a millimetre), piezoelectric materials have been adapted to an impressive range of applications that require only small amounts of displacement (typically less than a few thousandths of an inch). Today, piezoelectric ceramics are used in sensing applications, such as accelerometers, sensors, flow meters, level detectors, and hydrophones as well as in force or displacement sensors. The inverse piezoelectric effect is used in actuation applications, such as in motors and devices that precisely control position, and in generating sonic and ultrasonic signals.

Piezoelectric actuators can be used for the conversion of electrical energy to mechanical movement, for accurate positioning down to nanometer levels, for producing ultrasonic energy and sonar signals, and for the conversion of pressure and vibration into electrical energy. Piezoelectric actuators can also be manufactured in a variety of configurations and by a range of fabrication techniques. The industry recognizes these devices as monomorphs, bimorphs, stacks, co-fired actuators, and flexure elements. Piezoelectric actuators are found in telephones, stereo music systems, and musical instruments such as guitars and drums.

Piezoelectric actuators are beginning to appear in endoscope lenses used in medical treatment, and they are also used for valves in drilling equipment in offshore oil fields. They are also used to control hydraulic valves, act as small-volume pumps or special-purpose motors, and in other applications. Piezoelectric actuators that combine a number of superior characteristics will doubtless continue to evolve into powerful devices that support our society in the future.

(1) Operating principle

In 1880, Jacques and Pierre Curie discovered an unusual characteristic of certain crystalline minerals; when subjected to a mechanical force, the crystals became electrically polarized. Tension and compression generated voltages of opposite polarity, and in proportion to the applied force. Subsequently, the converse of this relationship was confirmed; if one of these voltage-generating crystals was exposed to an electric field it lengthened or shortened according to the polarity of the field, and in proportion to its strength. These behaviors were called the piezoelectric effect and the inverse piezoelectric effect, respectively.

The findings by Jacques and Pierre Curie have been confirmed again and again since then. Many polymers, ceramics, and molecules such as water are permanently polarized; some parts of the molecule are positively charged, and others are negatively charged. This behavior of piezoelectric materials is depicted in Figure 3.19(a). When the material changes dimensions as a result of an imposed mechanical force, a permanently polarized material such as quartz (SiO_2) or barium titanate ($BaTiO_3$) will produce an electric field. This behavior of piezoelectric materials when subject to an imposed force is depicted in Figure 3.19(b). Furthermore, when an electric field is applied to these materials, the polarized molecules within them will align themselves with the electric field, resulting

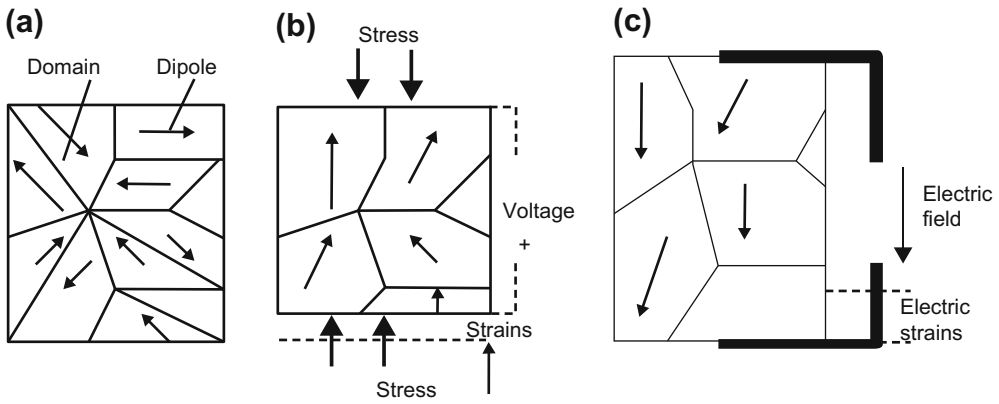


FIGURE 3.19

Behaviors of piezoelectric materials: (a) nonpolarized state when no force and no electricity are applied, (b) polarized state when compression stresses are imposed, and (c) polarized state when an electric field is applied after polarizing.

in induced dipoles within the molecular or crystal structure of the material, as illustrated in Figure 3.19(c).

Piezoelectricity involves the interaction between the electrical and mechanical behaviors of the material. Static linear relations between two electrical and mechanical variables are approximated by this interaction:

$$S = S^E T + dE,$$

$$D = dT + e^T E,$$

where S is the strain tensor, T is the stress tensor, E is an electric field vector, D is the electric displacement vector, S^E is the elastic compliance matrix when subjected to a constant electric field (the superscript E denotes that the electric field is constant), d is the matrix of piezoelectric constants, and e^T is the permittivity measured at constant stress.

The piezoelectric effect is, however, very nonlinear in nature. Piezoelectric materials exhibit, for example, strong hysteresis and drift that is not included in the above model. It should be noted, too, that the dynamics of the material are not described by the two equations above.

(2) Basic types

Piezoelectric devices make use of direct and inverse piezoelectric effects to perform a function. Some materials both these effects. For example, ceramics acquire a charge when being compressed, twisted or distorted, and produced physical displacements when electric voltages are imposed, several types of devices are available, some of the most important of which are listed here.

(1) Piezoelectric actuators

Piezoelectric actuators are devices that produce a small displacement with a high force capability when voltage is applied. There are many applications where a piezoelectric actuator may be used, such as ultra-precise positioning and in the generation and handling of high forces or pressures in static or dynamic situations.

Actuator configuration can vary greatly, depending on application. Piezoelectric stack or multilayer actuators are manufactured by stacking up piezoelectric disks or plates, the axis of the stack being the axis of linear motion that occurs when a voltage is applied. Tube actuators are monolithic devices that contract laterally and longitudinally when a voltage is applied between the inner and outer electrodes. A disk actuator is a device in the shape of a planar disk. Ring actuators are disk actuators with a center bore, making the actuator axis accessible for optical, mechanical, or electrical purposes. Other less common configurations include block, disk, bender, and bimorph styles.

These devices can also be ultrasonic. Ultrasonic actuators are specifically designed to produce strokes of several micrometers at ultrasonic ($>$20 kHz) frequencies. They are especially useful for controlling vibration, positioning applications, and quick switching. In addition, piezoelectric actuators can be either direct or amplified. The effect of amplification is not only larger displacement, but it can also result in slower response times.

The critical specifications for piezoelectric actuators are displacement, force, and operating voltage of the actuator. Other factors to consider are stiffness, resonant frequency, and capacitance. Stiffness is a term used to describe the force needed to achieve a certain deformation of a structure. For piezoelectric actuators, it is the force needed to elongate the device by a certain amount, normally specified in terms of Newtons per micrometer. Resonance is the frequency at which the actuators respond with maximum output amplitude. The capacitance is a function of the excitation voltage frequency.

(2) Piezoelectric motors

Piezoelectric motors use a piezoelectric, ceramic element to produce ultrasonic vibrations of an appropriate type in a stator structure. The elliptical movements of the stator are converted into the movement of a slider which is pressed into frictional contact with the stator. The consequent movement may either be rotational or linear depending on the design of the structure. Linear piezoelectric motors typically offer one degree of freedom, such as in linear stages, but they can be combined to provide more complex positioning factors. Rotating piezoelectric motors are commonly used in submicrometer positioning devices. Large mechanical torque can be achieved by combining several of these rotational units.

Piezoelectric motors have a number of potential advantages over conventional electromagnetic motors. They are generally small and compact for their power output, and provide greater force and torque than their dimensions would seem to indicate. In addition to a very positive size to power ratio, piezoelectric motors have high holding torque maintained at zero input power, and they offer low inertia from their rotors, providing rapid start and stop characteristics. Additionally, they are unaffected by electromagnetic fields, which can hamper other motor types. Piezoelectric motors usually do not produce magnetic fields and also are not affected by external magnetic fields. Because they operate at ultrasonic frequencies, these motors do not produce sound during operation.

(3) Multilayer piezoelectric benders

High efficiency, low-voltage, multilayer benders have been developed to meet the growing demand for precise, controllable, and repeatable deflection devices in the millimetre and micrometer range. Multilayer, piezoelectric, ceramic benders are devices capable of rapid ($<$ 10 ms) millimetre movements with micrometer precision. They utilize the inverse piezoelectric effect, in which an electric field creates a cantilever bending effect. By making the ceramic layers very thin, between 20 and 40 μm, deflections can be generated with low power consumption at operating voltages from -10 to $+60$ V. With an electrical field of <3 kV/mm, large deflections per unit volume can be achieved with high reliability.

(4) Piezoelectric drivers and piezoelectric amplifiers

Piezoelectric drivers and piezoelectric amplifiers have been developed to match the requirements for driving and controlling piezoelectric actuators and stages in some applications. Standard linear amplifier products are simple voltage followers that amplify a low-voltage input signal, and others are recommended for use as integrated or stand-alone systems in applications requireing more advanced capabilities for closed-loop servo control. A voltage amplifier is typically needed to control piezoelectric actuators due to the high operating voltage needed for the former. In other words, before the computer, through a direct to alternating converter, provides the control signal, it must be amplified.

Piezoelectric actuators theoretically have an unlimited resolution. Therefore, every infinitely small voltage step caused for example, by the noise of the amplifier, is transformed into an infinitely small mechanical shift. The noise characteristics of the amplifier are very important in design of a precision positioning system.

Problems

1. For those machines you are familiar with, such as washing machines and multifunctional copiers, plot the hierarchical levels of the control systems inside them, based on the definition given in Figure 3.1.
2. Analyze the working principle that computers use to adjust their screen brightness and contrasts; think about designing a color sensor for a computer to implement these functions.
3. Please explain why one requirement for the utilization of the laser light section method is an at least partially diffuse reflecting surface.
4. Please explain why, to ensure widely constant signal amplitude on the laser light section sensor, the depth of focus of the camera lens as well as the depth of focus of the laser line generator has to cover the complete measurement elevation range.
5. Please list all the similarities and differences between color sensors and scan sensors.
6. Should bimetallic sensors be classified as contact or noncontact temperature sensors? Should ultrasonic distance sensors be classified as passive or active physical sensors?
7. Please explain why a contact temperature sensor requires that both the sensor and the object in measurement are in thermal equilibrium, that is, there is no heat flow between them.
8. Please find the respective spectral region of infrared light, red light, and laser light for the optical distance sensors given in Table 3.2.
9. Based on Table 3.3, try to give a definition of measurement sensors.
10. In additional to the piezoelectric principle, both force and load sensors can work by photoelectric principles. Please explain how photoelectric physics enables the working of force and load sensors.

11. The three elementary components of an electric actuator are gears, motor, and switch. However, the architecture of an electric actuator shown in Figure 3.12 gives one more component in addition to these three elementary components; the hand wheel. Is this hand wheel necessary for an electric actuator?
12. As shown in Figure 3.13, the Hall voltage, V_H, can be used to sense the changes of the magnetic field through the Hall plate, which is the working principle of the Hall sensor. Do you think that Hall effect can be used to make a magnetic actuator?
13. Please plot the architecture diagram for a hydraulic jack, as, for example, is used in garages.
14. Please figure out the elementary components of a pneumatic actuator.
15. Can you give the similarities and differences between pneumatic and hydraulic actuators?

Further Reading

LabAutoPedia (www.labautopedia.com). Sensors. http://www.labautopedia.com/mw/index.php/Sensors. Accessed: May 2009.

Temperatures (www.temperatures.com). Temperature sensors. http://www.temperatures.com/index.html. Accessed: May 2009.

GlobalSpec (www.globalspec.com). Distance sensors. http://search.globalspec.com/ProductFinder/FindProducts?query=distance%20sensor. Accessed: May 2009.

GlobalSpec (www.globalspec.com). Color sensors. http://search.globalspec.com/productfinder/findproducts?query = color%20sensor. Accessed: Mayl 2009.

Pepperl + Fuchs (www.am.pepperl fuchs.com). http://www.am.pepperl fuchs.com/products/productfamily.jsp?division= FA & productfamily id = 1455. Accessed: April 2008.

Pepperl + Fuchs (www.am.pepperl fuchs.com). http://www.am.pepperl fuchs.com/products/productfamily.jsp?division= FA & productfamily id= 1575. Accessed: April 2008.

Sensors (www.sensorsmag.com). http://www.sensorsmag.com/articles/1298/mag1298/main.shtml. Accessed: April 2008.

Sensors (www.sensorsmag.com). http://www.sensorsmag.com/sensors/article/articleDetail.jsp?id = 179165. Accessed: May 2008.

Short Courses (www.shortcourses.com). http://www.shortcourses.com/choosing/sensors/05.htm. Accessed: April 2008.

Sick (www.sick.com). http://www.sick.com/home/factory/catalogues/industrial/colorsensors/en.html. Accessed: April 2008.

Sick (www.sick.com). http://englisch.meyle.de/contract partner/sick magnetic proximity sensors.php. Accessed: April 2008.

Siemens (www.sbt.siemens.com). 2005. Actuators and valves. http://www.sbt.siemens.com/hvp/components/products/damperactuators/default.asp. Accessed: June 2008.

Dytran (www.dytran.com). Introduction to piezoelectric force sensors. http://www.dytran.com/graphics/a4.pdf. Accessed: May 2009.

Direct Industry (http://www.directindustry.com). http://www.directindustry.com/industrial manufacturer/magnetic sensor 70932.html. Accessed: April 2009.

Direct Industry (http://www.directindustry.com). http://www.directindustry.com/industrial manufacturer/magnetic switch 72332.html. Accessed: April 2009.

Direct Industry (http://www.directindustry.com). http://www.directindustry.com/industrial manufacturer/hall effect sensor 71708.html. Accessed: April 2009.

Direct Industry (http://www.directindustry.com). http://www.directindustry.com/industrial manufacturer/reed switch 74782.html. Accessed: April 2009.
Hydraulic Tutorial (http://www.hydraulicsupermarket.com). http://www.hydraulicsupermarket.com/technical.html. Accessed: May 2009.
PAControl (http://electricalequipment.pacontrol.com). http://electricalequipment.pacontrol.com/proximitysensors.html. Accessed: April 2009.
PAControl (http://electricalequipment.pacontrol.com). ttp://electricalequipment.pacontrol.com/capacitiveproximitysensors.html. Accessed: April 2009.
PAControl (http://electricalequipment.pacontrol.com). ttp://electricalequipment.pacontrol.com/inductiveproximitysensors.html. Accessed: April 2009.
PAControl (http://electricalequipment.pacontrol.com). http://electricalequipment.pacontrol.com/magnetic proximitysensors.html. Accessed: April 2009.
Physics psu.edu. http://class.phys.psu.edu/p457/experiments/html/hall effect 2004.htm . Accessed: April 2009.
PI (http://www.physikinstrumente.com). http://www.physikinstrumente.com/en/products/piezo tutorial.php. Accessed: May 2009.
Piezo (http://www.piezo.com). http://www.piezo.com/tech2intropiezotrans.html. Accessed: May 2009.

CHAPTER

Transducers and valves

4

A transducer is a device, usually electrical, electronic, electro-mechanical, electromagnetic, photonic or photovoltaic, that converts input physical energy in one form into output physical energy of another form for various purposes, including measurement or information transfer. A typical example of a transducer is a recorder, which converts sound (mechanical) energy into electrical (electromagnetic) energy.

In such devices an electrical switch opens and closes under the control of another electrical or electronic circuit. Earlier devices had a switch that is operated by an electromagnet to open or close one, or many sets of contacts to form a high-voltage circuit controlled by a low-voltage signal, as in some types of modems or audio amplifiers. However, as a field device in an industrial control system, such a switch will be a device of two states, On and Off, the value of which being dependent on some controls or conditions.

A valve is a device that regulates the flow of materials (gases, fluidized solids, slurries, or liquids) by opening, closing, or partially obstructing various passageways. Valves are technically pipe fittings, but are usually considered separately. They are used in a variety of applications, including oil and gas, power generation, mining, water reticulation, sewerage and chemical manufacturing.

4.1 INDUSTRIAL SWITCHES

4.1.1 Limit switches

A limit switch is an electromechanical device that can be used to determine the physical position of equipment. Its primary purpose is to control the intermediate or end limits of a linear or rotary motion. For example, an extension on a valve shaft mechanically trips a limit switch as it moves from open to shut, or from shut to open. The limit switch is designed to give a signal to an industrial control system when a moving component such as an overhead door or piece of machinery has reached the limit (end point) of its travel, or just a specific point on its journey. It is often used as a safety device to protect against accidental damage to equipment.

(1) Operating principle

A linear limit switch is an electromechanical device that requires physical contact between an object and the activator of the switch to make the contacts change state. As an object (target) makes contact with the actuator of the switch, it moves the activator to the limit where the contacts change state.

Limit switches can be used in almost any industrial environment, because of their typically rugged design. However, the device uses mechanical parts that can wear over time and it is slow when compared to noncontact, electrical devices such as proximity or photoelectric sensors.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10004-X

Rotary limit switches are similar to relays, in that they are used to allow or prevent current flow when in the closed or open position. The groupings within this family are usually defined by the manner in which the switch is actuated (e.g., rocker, foot, read, lever, etc.). Rotary limit switches can range from simple push-button devices, usually used to delineate between ON and OFF, to rotary and toggle devices for varying levels, through to multiple-entry keypads, for multiple control functions. In addition to maintaining or interrupting flow, and maintaining flow levels, switches are used in safety applications as security devices (locker switches) and as functionary actuators when controlled by sensors or computer systems.

Normally, the limit switch gives an ON/OFF output that corresponds to a valve position. Limit switches are used to provide full open or full shut indications as illustrated in Figure 4.1, which gives a typical linear limit switch operation. Many limit switches are of the push-button variety. In Figure 4.1, when the valve extension comes into contact with the limit switch, the switch depresses to complete, or turn on, an electrical circuit. As the valve extension moves away from the limit switches, spring pressure opens the switch, turning off the circuit. Limit switch failures are normally mechanical in nature. If the proper indication or control function is not achieved, the limit switch is probably faulty. In this case, local position indication should be used to verify equipment position.

(2) Basic types

The basic types of limit switches are: linear limit switches, where an object will move a lever (or simply depress a plunger) on the switch far enough for the contact in the switch to change state; rotary limit switches, where a shaft must turn a preset number of revolutions before the contact changes state, as used in cranes, overhead doors, etc.; and magnetic limit switches, or reed switches, where the object is not touched but sensed. Table 4.1 gives the details of the classes and types of limit switches.

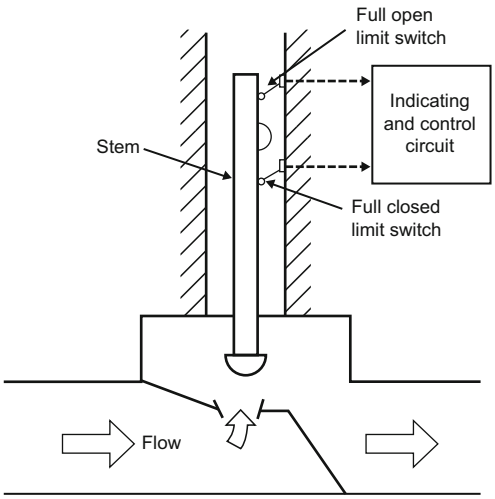


FIGURE 4.1

Operations of a linear limit switch.

Table 4.1 Classes and Types of Limit Switches

Class	Type	Descriptions
Linear limit switches	Safety limit switches	Safety limit switches are designed for use with moveable machine guards/access gates, which must be closed for operator safety and for any other presence, and position-sensing applications normally addressed with conventional limit switches. Their positive opening network connection contacts provide a higher degree of reliability than conventional spring-driven switches whose contacts can weld or stick shut. These limit switches are of multiple actuator styles and four 90° head positions so that they provide application versatility.
	Comprehensive range limit switches	Comprehensive range limit switches are provided in the most popular industrial sizes, shapes, contact configurations, as well as an innovative series of encapsulated limit switches, available with a connection cable or plug. Their output contacts are offered in snap action, slow action, or overlapping configurations. Various types of push button and roller actuators, roller levers, and multidirectional levers are available. These switches are ideal for manufacturers of material handling, packaging, conveying, and machine tool equipment.
	Mechanical limit switches	Mechanical limit switches are frequently used in position detection on doors and machinery and parts detection on conveyors and assembly lines. Their range comprises various housing and actuator styles, choice of slow or fast action contacts, and various contact arrangements.
Rotary limit switches	Single-deck rotary switches	Single-deck rotary switches can control several circuits at a time. Actuator choices for single-deck rotary switches include flush actuator, bare shaft actuator, knobbed shaft, and locker. In a flush actuator configuration the actuator does not project above the switch body. Typically it requires a screwdriver for operation. In a bare shaft actuator configuration the shaft has no knob, but may be notched to accept various knob configurations. A knobbed shaft comes with an integral knob. In a locker configuration the actuation is done with a key or other security or tamperproof method.
	Multiple-deck rotary switch	Multiple-deck rotary switches can control several circuits simultaneously. Actuator choices for multiple-deck rotary switches include flush actuator, bare shaft actuator, knobbed shaft, and locker. In a flush actuator, the actuator does not project above the switch body, and typically requires a screwdriver for operation. In a bare shaft actuator, the shaft has no knob, but may be notched to accept various knob configurations. A knobbed shaft comes with an integral knob. In a locker, the actuation is done with a key or other secure or tamperproof method.
	Magnetic limit switches	Magnetic limit switches work based on the operation of reed switches, and are more reliable than the reed switches because of their simplified construction. The switches are constructed of flexible ferrous strips (reeds) and are placed near the intended travel of the valve stem or control rod extension.

Limit switches can differ in terms of their working and contact mechanisms. There are two types of switch working mechanisms; electromechanical and solid-state. Electromechanical limit switches have mechanical contacts, relays, or reeds. Solid-state limit switches are electronic and do not have moving parts. There are three choices for contacts; momentary contact, maintained contact, and positive opening. Momentary contact means that the switch is open or closed only during actuation. Maintained contact means that the limit switch contacts remain in the triggered position even after the actuator has been released, and are reset only by further mechanical action of the operating head. Positive opening means that the contact-point opens reliably and then remains open, in the activated position, even in the event of a mechanical failure.

Limit switches can also differ in terms of orientation and performance. The orientation of the limit switch actuator is critical when determining sizing requirements. Some limit switches have a top-mounted actuator, whilst others have the actuator on the side of the limit switch. In terms of performance, limit switches carry specifications for maximum current rating, maximum AC (alternating current) voltage rating, maximum DC (direct current) voltage rating, minimum mechanical life, minimum electrical life, and operating temperature. Some limit switches are designed for use in transistor-transistor logic (TTL) circuits.

4.1.2 Photoelectric switches

Photoelectric switches represent perhaps the largest variety of problem-solving choices in the field of industrial control systems. Today, photoelectric technology has advanced to the point where it is common to find a sensor or a switch that can detect a target less than 1 millimetre in diameter, while other units have a sensing range up to 60 meters.

A very familiar application of photoelectric switches is to turn a trap on at dark and off at daylight. This enables the user to set the trap out earlier and retrieve it later in the morning while reducing wear on the battery. Quite often, a garage door opener has a through-beam photoelectric switch mounted near the floor, across the width of the door, which makes sure nothing is in the path of the door when it is closing. A more industrial application for a photoelectric device is the detection objects on a conveyor. An object will be detected any place on a conveyor running between the emitter and the receiver, as long as there is a gap between the objects and the switch's light does not "burn through" the object. This term refers to an object that is thin or light in color, and so allows the light from the emitter to the target pass through, hence the receiver never detects the object.

Almost all photoelectric switches contain three components: an emitter, which is a light source such as a light-emitting diode or laser diode, a photodiode; a phototransistor receiver to detect the light source; as well as the supporting electronics designed to amplify the signal relayed from the receiver. Photoelectric sensing uses a beam of light to detect the presence or absence of an object; the emitter transmits a beam of light, either visible or infrared, which is directed to and detected by the receiver.

All photoelectric sensors and switches identify their output as "dark-on" and "light-on," which refers to the output of the switch if the light source hits the receiver or not. If an output is present while no light is received, this would be called a "dark-on" output. In reverse, if the output is "on" while the receiver detects the light from the emitter, the sensor or switch would have a "light-on" output. The types of output needs to be selected prior to purchasing the sensor, unless it is user-adjustable, in which case it can be decided upon during installation, by either flipping a switch or wiring the sensor accordingly.

Photoelectric configurations are categorized by the method in which light is emitted and delivered to the receiver. There are three main categories; namely through beam, retroreflective, and proximity.

(1) Through beam photoelectric switches are configured by placing the emitter and detector opposite the path of the target and presence is sensed when the beam is broken.

(2) Retroreflective photoelectric switches are configured with the emitter and detector in the same housing and rely on a reflector to bounce the beam back across the path of the target. This type may be polarized to minimize false reflections.

(3) Proximity photoelectric switches have the emitter and detector in the same housing and rely upon reflection from the surface of the target. This mode can include presence sensing and distance measurement via analog output.

The proximity category can be further broken down into five submodes; diffuse, divergent, convergent, fixed-field, and adjustable-field. With a diffuse switch the presence of an object is detected when any portion of the diffusely reflected signal bounces back from the detected object. Divergent beam switches are short-range, diffuse-type switches without collimating lenses.

Convergent, fixed-focus, or fixed-distance optics (such as lenses) are used to focus the emitter beam at a fixed distance from the sensor or switch. Fixed-field switches are designed to have a distance limit beyond which they will not detect objects, no matter how reflective. Adjustable-field switches utilize a cut-off distance beyond which a target will not be detected, even if it is more reflective than the target. Some photoelectric switches can be set to different optical sensing modes. The reflective properties of the target and environment are important considerations in the choice and use of photoelectric switches.

Diffuse photoelectric switches operate under a somewhat different style to retroreflective and through-beams, although the operating principle remains the same. Diffuse photoelectric switches actually use the target as the reflector, such that detection occurs upon reflection of the light from the object back onto the receiver, as opposed to using an interruption of an emitted light beam. The emitter sends out a beam of light, often a pulsed infrared, visible red, or laser beam, which is reflected by the target when it enters the detectable area. Diffuse reflection bounces light off the target in all directions. Part of the beam will return to the receiver that it was emitted from inside the same housing which is locate. Detection occurs, and the output will either turn on or off (depending upon whether it is light-on or dark-on) when sufficient light is reflected back to the receiver. This can be commonly witnessed in airport washrooms, where a diffuse photo electric switch will detect your hands as they are placed under the faucet and the attending output will turn the water on. In this application, your hands act as the reflector.

To ensure repeatability and reliability, photoelectric switches are available with three different types of operating principles; fixed-field sensing, adjustable-field sensing, and background suppression through triangulation. In the simplest terms, these switches are focused on a specific point in the foreground, ignoring anything beyond that point.

(1) Standard fixed-field switches operate optimally at their preset “sweet spot”; the distance at which the foreground receiver will detect the target. As a result, these switches must be mounted within a certain fixed distance of the target. In fixed-field technology, when the emitter sends out a beam of light, two receivers sense the light on its return. The short-range receiver is focused on the target object’s location. The long-range receiver is focused on the background. If the long-range receiver detects a higher intensity of reflected light than the short-range receiver, the output will not

turn on. If the short-range receiver detects a higher intensity of reflected light than the long-range receiver, an output occurs and the object is detected.

(2) Adjustable-field switches operate under the same principle as fixed-field switches, but the sensitivity of the receivers can be electrically adjusted, by using a potentiometer. By adjusting the level of light needed to trigger an output, the range and sensitivity of the switch can be altered to fit the application.

(3) Background suppression by triangulation also emits a beam of light that is reflected back to the switch. Unlike fixed- and adjustable-field switches, which rely on the intensity of the light reflected back to them, background suppression sensors rely completely on the angle at which the beam of light returns. Like fixed- and adjustable-field switches, background suppression switches feature short-range and long-range receivers in fixed positions. In addition, background suppression sensors or switches have a pair of lenses that are mechanically adjusted to focus the reflected beam precisely to the appropriate receiver, changing the angle of the light received. The long-range receiver is focused through the lens on the background. Deflected light returning along that focal plane will not trigger an output. The short-range receiver is focused, through a second lens, on the target. Any deflected light returning along that focal plane will trigger an output; an object will be detected.

4.1.3 Proximity switches

Proximity sensing is the technique of detecting the presence or absence of an object by using a critical distance. Proximity sensors detect the presence of an object without physical contact, and proximity switches execute the necessary responses when sensing the presence of the target. A position sensor determines an object's coordinates (linear or angular) with respect to a reference; displacement means moving from one position to another for a specified distance (or angle). In effect, a proximity sensor is a threshold version of a position sensor.

Typical applications include the control, detection, position, inspection, and automation of machine tools and manufacturing systems. They are also used in packaging, production, printing, plastic merging, metal working, and food processing, etc. The measurement of proximity, position, and displacement of objects is essential in determining valve position, level detection, process control, machine control, security, etc. Special-purpose proximity sensors perform in extreme environments (exposure to high temperatures or harsh chemicals), and address specific needs in automotive and welding applications. Inductive proximity sensors are ideal for virtually all metal-sensing applications, including detecting all metals or, nonferrous metals selectively.

Proximity sensors and switches can have one of many physica and technology types. The physics types of proximity sensors and switches include capacitive, inductive, photoelectric, ultrasonic, and magnetic. Common terms for technology types of proximity sensors and switches include eddy current, air, capacitance, infrared, fiber optics, etc. Proximity sensors and switches can be contact or noncontact.

(1) Physics of different types of proximity sensors and switches

(a) Capacitive proximity sensors and switches

Capacitive sensing devices utilize the face or surface of the sensor as one plate of a capacitor, and the surface of a conductive or dielectric target object as the other. Capacitive proximity sensors can be a sensor element or chip, a sensor or transducer, an instrument or meter, a gauge or indicator,

a recorder or totalizer, or a controller. Common body styles for capacitive proximity sensors are barrel, limit switch, rectangular, slot style, and ring. A barrel body style is cylindrical in shape, typically threaded. A limit switch body style is similar in appearance to a contact limit switch. The sensor is separated from the switching mechanism and provides a limit-of-travel detection signal. A rectangular or block body style is a one-piece rectangular or block-shaped sensor. A slot style body is designed to detect the presence of a vane or tab as it passes through a sensing slot, or U channel. A ring-shaped body style is a doughnut-shaped sensor, where the object passes through the center of the ring.

(b) Inductive proximity sensors and switches

Inductive proximity sensors are noncontact proximity devices that set up a radio frequency field by using an oscillator and a coil. The presence of an object alters this field, and the sensor is able to detect this alteration. The body style of inductive proximity sensors can be barrel, limit switch, rectangular, slot, or ring.

(c) Photoelectric proximity sensors and switches

These sensors utilize photoelectric emitters and receivers to detect distance, presence, or absence of target objects. Proximity photoelectric sensors have the emitter and detector in the same housing and rely upon reflection from the surface of the target. Some photoelectric sensors can be set for multiple different optical sensing modes, including presence sensing and distance measurement via analog output. The proximity category can be further broken down into five submodes; diffuse, divergent, convergent, fixed-field, and adjustable-field. A diffuse sensor senses presence when any portion of the diffuse reflected signal bounces back from the detected object. Divergent beam sensors are short-range, diffuse-type sensors without any collimating lenses. Convergent, fixed-focus, or fixed-distance optics (such as lenses) are used to focus the emitter beam at a fixed distance from the sensor. Fixed-field sensors are designed to have a distance limit beyond which they will not detect objects, no matter how reflective. Adjustable-field sensors utilize a cut-off distance beyond which a target will not be detected, even if it is more reflective than the target.

(d) Ultrasonic proximity sensors and switches

Ultrasonic proximity sensing can use a sensor element or chip, a sensor or transducer, an instrument or meter, a gauge or indicator, a recorder or totalizer, or a controller. The body style of the ultrasonic proximity sensors can be barrel, limit switch, rectangular, slot, or ring.

(e) Magnetic proximity sensors and switches

These noncontact proximity devices utilize inductance, Hall effect principles, variable reluctance, or magnetoresistive technology. Magnetic proximity sensors are characterized by the possibility of large switching distances available from sensors with small dimensions. Magnetic objects (usually permanent magnets) are used to trigger the switching process. As the magnetic fields are able to pass through many nonmagnetic materials, the switching process can be triggered without the need for direct exposure to the target object. By using magnetic conductors (e.g., iron), the magnetic field can be transmitted over greater distances so that, for example, the signal can be carried away from high-temperature areas.

(2) Technical types of proximity sensors and switches

(a) Eddy current proximity sensor or switch

In an eddy current proximity sensor, electrical currents are generated in a conductive material by an induced magnetic field. Interruptions in the flow of these eddy currents, which are caused by imperfections or changes in a material's conductive properties, will cause changes in the induced magnetic field. These changes, when detected, indicate the presence of change in the test object. Eddy current proximity sensors and switches detect the proximity or presence of a target by sensing the magnetic fields generated by a reference coil. They also measure variations in the field due to the presence of nearby conductive objects. Field generation and detection information is provided in the kilohertz to the megahertz range. They can be used as proximity sensors to detect presence of a target, or they can be configured to measure the position or displacement of a target.

(b) Air proximity sensor or switch

The air proximity sensor is a noncontact, no-moving-part sensor. In the absence of an object, air flows freely from the sensor, resulting in a near zero output signal. The presence of an object within the sensing range deflects the normal air flow and results in a positive output signal. At low supply pressure, flow from the sensor exerts only minute forces on the object being sensed and is consequently appropriate for use where the target is light-weight or easily marred by mechanical sensors.

(c) Capacitance proximity sensor or switch

Many industrial capacitance sensors or switches work by means of the physics of capacitance. This states that the capacitance varies inversely with the distance between capacitor plates. In this arrangement a certain value can be set to trigger target detection. Note that the capacitance is proportional to the plate area, but is inversely proportional to the distance between the plates. When the plates are close to each other, even a small change in distance between them can result in a sizeable change in capacitance. Some of the capacitance proximity switches are tiny one-inch cube electronic modules that operate using a capacitance change technique. These sensors or switches contain two switch outputs a latched output, which toggles the output ON and OFF with each cap input activation, and a momentary output, which will remain activated as long as the sensor input capacitance is higher than the level set by the module's adjustment screw.

(d) Infrared proximity sensor or switch

Infrared light is beyond the light range that is visible to the human eye, and falls between the visible light and microwave regions (the wavelength is longer than visible light). The longest wavelengths are red, which is where infrared got its name ("beyond red"). Infrared waves are electromagnetic waves, which is also present in heat; heat from campfires, sunlight, etc., is actually infrared radiation. Infrared proximity sensors work by sending out a beam of infrared light, and then computing the distance to any nearby objects by employing the characteristics of the returned signal.

(e) Fiber-optic proximity sensor or switch

Fiber-optic proximity sensors are used to detect the proximity of target objects. Light is supplied and returned via glass fiber-optic cables. These can fit in small spaces, are not susceptible to electrical noise, and have no risk of sparking or shorting. Glass fiber has very good optical qualities and good

high-temperature ratings. Plastic fiber can be cut to length in the field and can be flexible enough to accommodate various routing configurations.

4.2 INDUSTRIAL TRANSDUCERS

4.2.1 Ultrasonic transducers

Sound that is generated beyond the level of human hearing range is called ultrasound. Although ultrasound typically starts at 20 kHz, most ultrasonic transducers start at 200 kHz. Ultrasound, which is similar in nature to audible sound, has far shorter wavelengths and is suitable for detecting small flaws. These shorter wavelengths make ultrasound extremely useful for non-destructive testing and measurement of materials. An ultrasonic transducer itself is a device that is capable of generating and receiving ultrasonic vibrations.

An ultrasonic transducer is made up of an active element, a backing, and a wear plate. The active element is a piezoelectric or single crystal material, which converts electrical energy to ultrasonic energy. It also receives back ultrasonic energy and converts this to electrical energy. The electrical energy pulse is generated in an instrument such as a flaw detector.

The sound field of an ultrasonic transducer has two distinct zones, as indicated in Figure 4.2. These zones are called the near field, which is the region directly in front of the transducer, and the far field, which is the area beyond N where the sound field pressure gradually drops to zero. The near-field distance is a function of the transducer frequency, element diameter, and the sound velocity of the test material, as shown in the following equation:

(Figure 4.2 gives the definitions for $N, D,$ and C) $N = D^2/4c$.

There are several sound field parameters that are very useful in describing the characteristics of an ultrasonic transducer. Knowledge of the focal length, beam width and focal zone may be necessary in

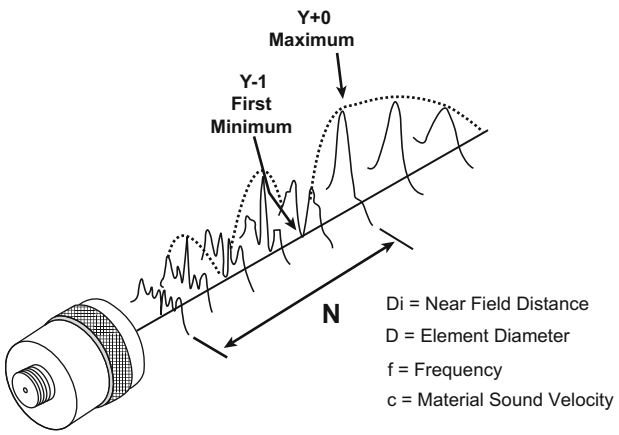


FIGURE 4.2

The sound field of ultrasonic transducers.

order to determine whether a particular transducer is appropriate for an application. The beam angle is the total included angle of ultrasonic beam. In general, a high-frequency transducer will produce a narrow beam and a lower-frequency transducer a wider beam.

All transducers have beam spread, which is important when inspecting flaws that may be close to certain geometric features of the material to be tested, including side walls and corners that may cause spurious echoes which could be mistaken for flaws or defects. For flat transducers the pulse-echo beam spread angle, a, is well defined and is given by the equation:

$$\text{Sine}\ (a/2) \ = \ 0.514c/fD.$$

It can be seen from this equation that beam spread in a transducer can be reduced by selecting a transducer with a higher frequency, or a larger diameter, or both.

Ultrasonic transducers are classified into two groups according to the application.

(1) Contact transducers

Contact transducers are used for direct contact inspections, and are generally hand manipulated. They have elements that are protected in a rugged casing to withstand sliding contact with a variety of materials. These transducers have an ergonomic design so that they are easy to grip and move along a surface. They often have replaceable wear plates to lengthen their useful life. Coupling materials of water, grease, oils, or commercial fluids are used to remove the air gap between the transducer and the component being inspected.

(2) Immersion transducers

Immersion transducers do not contact the component. These transducers are designed to operate in a liquid environment, and so all connections are watertight. Immersion transducers usually have an impedance matching layer that helps to get more sound energy into the water and, in turn, into the component being inspected. Immersion transducers can have a cylindrically or spherically focused lens, as shown in Figure 4.3. A focused transducer can improve its sensitivity and axial resolution by concentrating the sound energy to a smaller area. They are typically used inside a water tank, or as part of a squinter or bubbler system in scanning applications.

Contact transducers are available in a variety of configurations to improve their adaptability. Three examples are detailed below.

(1) Dual-element transducers contain two, independently operated elements in a single housing. One of the elements transmits and the other receives the ultrasonic signal. These two elements are angled towards each other to create a crossed-beam sound path in the test material. Figure 4.4(a) shows the operating principle of dual-element transducers. Active elements can be chosen for their transmission and receiving capabilities to provide a transducer with a cleaner signal, or designed for special applications, such as the inspection of coarse-grained material. Dual-element transducers are especially well suited for making measurements in applications where reflectors are very near the transducer, since this design eliminates the ring-down effect that single-element transducers experience.

(2) Delay line transducers provide versatility by having a variety of replaceable options. As shown in Figure 4.4(b), delay line transducers have a removable delay line, a surface conforming membrane, and protective wear cap options. These components can make a single transducer effective for a wide range of applications. The primary function of a delay line transducer is to introduce a time delay

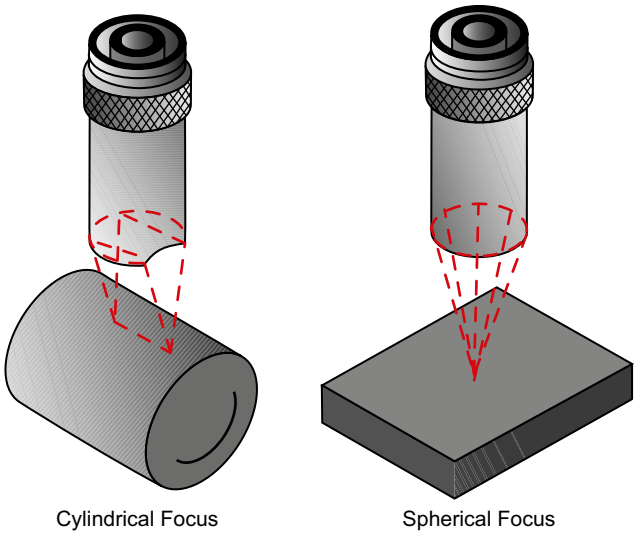


FIGURE 4.3

Two types of focuses used in immersion ultrasonic transducers.

between the generation of the sound wave and the arrival of any reflected waves. This allows the transducer to complete its sending function before it starts its listening function, so that near surface resolution is improved. They are designed for use in applications such as high-precision thickness gauging of thin materials and delaminating checks in composite materials. They are also useful in high-temperature measurement applications, since the delay line provides some insulation from heat to the piezoelectric element.

(3) Angle beam transducers are typically used to introduce a refracted shear wave into the test material. Transducers can use a variety of fixed angles, or in adjustable versions where the user determines the angles of incidence and refraction. In the fixed-angle versions, the angle of refraction that is marked on the transducer is only accurate for a particular material, which is usually steel. The angled sound path allows the sound beam to be reflected from the back wall of the target to improve detectability of flaws in and around welded areas. They are also used to generate surface waves for use in detecting defects on the surface of a component. Figure 4.4(c) shows the operating principle of angle beam transducers.

4.2.2 Linear and rotary variable differential transformers

The linear variable differential transformer (LVDT) is a well-established transducer design which has been used for many decades for the accurate measurement of displacement, and within closed loops for the control of positioning. The rotational variable differential transformer (RVDT) is also a well-established transducer design used to measure rotational angles and operates under the same principles as the LVDT sensor. Whilst the LVDT has a cylindrical iron core, the RVDT has a rotary ferromagnetic core.

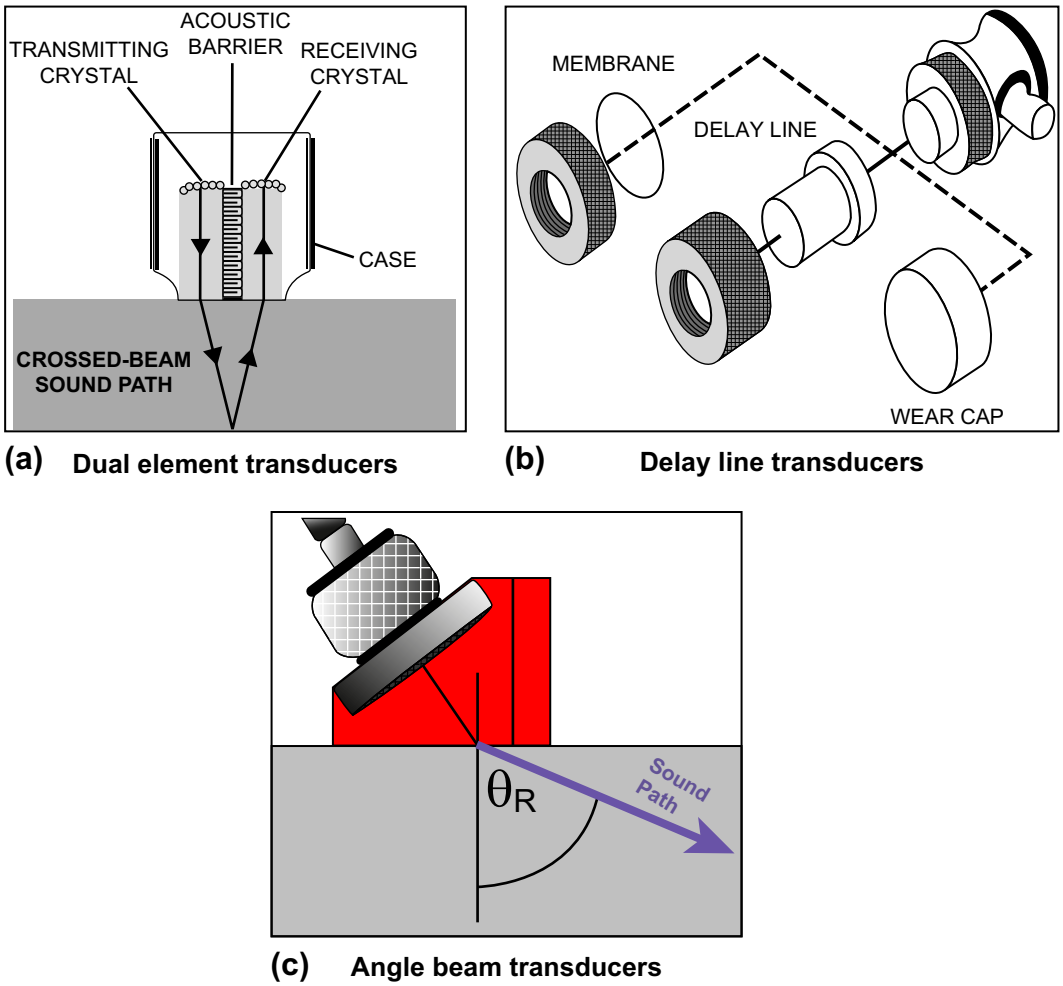


FIGURE 4.4

Three types of special contact ultrasonic transducers.

(Courtesy of NDT Resource Centre, 2009.)

Both the LVDT and RVDT designs lend themselves to easy modification in order to fulfil a whole range of different applications in both research and industry. This includes pressurized versions for hydraulic applications, materials suitable for sea water and marine services, dimensions to suit specific application requirements, multichannel, rack amplifier-based systems, automotive suspension systems.

(1) An LVDT is much like any other transformer, in that it consists of a primary coil, secondary coils, and a magnetic core as illustrated in Figure 4.5(a). The primary coils (the upper coil in Figure 4.5(a)) are energized with constant-amplitude alternating current. This produces an alternating

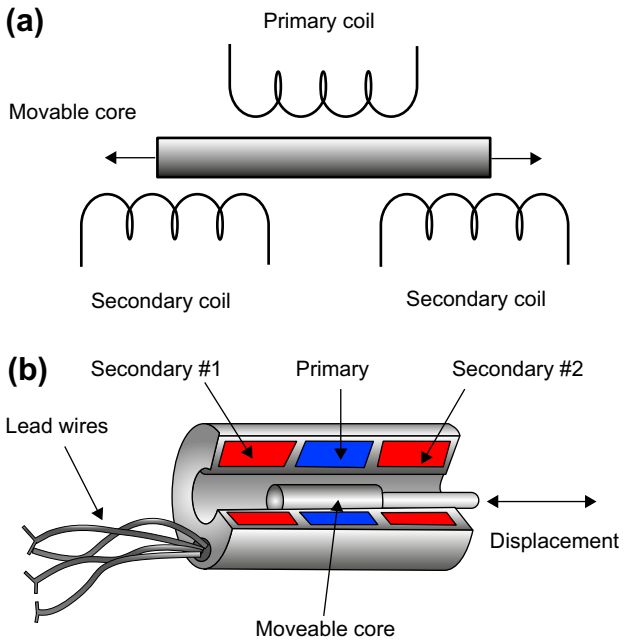


FIGURE 4.5

The working principles of an LVDT: (a) the three coils and the movable core and (b) the displacement system of a sensor.

magnetic field in the center of the transducer, which induces a signal into the secondary coils (the two lower coils in Figure 4.5(a)) depending on the position of the core.

Movement of the core within this area causes the secondary signal to change (Figure 4.5(b)). As the two secondary coils are positioned and connected in a set arrangement (push pull mode), when the core is positioned at the center, a zero signal is produced. Movement of the core from this point in either direction causes the signal to increase. Since the coils are wound in a particular precise configuration, the signal output has a linear relationship with the actual mechanical movement of the core.

The secondary output signal is then processed by a phase-sensitive demodulator which is switched at the same frequency as the primary energizing supply. This results in a final output which, after rectification and filtering, gives direct current output that is proportional to the core movement, and also indicates its direction, positive or negative, from the central zero point (Figure 4.5(b)).

As with any transformer, the voltage of the induced signal in the secondary coil is linearly related to the number of coils. The basic transformer relation is:

$$V_{out}/V_{in} = N_{out}/N_{in},$$

where V_{out} is the voltage at the output, V_{in} is the voltage at the input, N_{out} is the number of windings of the output coil, and N_{in} is the number of windings of the input coil.

The distinct advantage of using an LVDT displacement transducer is that the moving core does not make contact with other electrical components of the assembly, which does occur in resistive types,

and so offers high reliability and long life. Further, the core can be so aligned that an air gap exists around it, which is ideal for applications where minimum mechanical friction is required.

(2) An RVDT is an electromechanical transducer that provides a variable alternating current output voltage. This output voltage is linearly proportional to the angular displacement of its input shaft. When energized with a fixed alternating current source, the output signal is linear within a specified range of the angular displacement. RVDT utilizes brushless, noncontacting technology to ensure long life, reliability, and repeatable position sensing with infinite resolution. Such reliable and repeatable performance ensures accurate position sensing under the most extreme operating conditions.

As shown in Figure 4.6, rotating a ferromagnetic-core bearing that is supported within a housed stator assembly is the basis of RVDT construction and operation. The housing is stainless steel. The stator consists of a primary excitation coil and a pair of secondary output coils. A fixed, alternating current excitation is applied to the primary stator coil that is electromagnetically coupled to the secondary coils. This coupling is proportional to the angle of the input shaft. The output pair is structured so that one coil is in phase with the excitation coil, and the second is 180° out of phase with it. When the rotor is in a position that directs the available flux equally to both the in phase and out of phase coils, the output voltages cancel and result in a zero output signal. This is referred to as the electrical zero position. When the rotor shaft is displaced from the electrical zero position, the resulting output signals have a magnitude and phase relationship which is proportional to the direction of rotation.

Because the performance of an RVDT is essentially similar to that of a transformer, excitation voltage changes will produce directly proportional changes in the output (transformation ratio). However, the voltage out to excitation voltage ratio will remain constant. Since most RVDT signal conditioning systems measure signal as a function of the transformation ratio, excitation voltage drift beyond 7.5% typically has no effect on sensor accuracy and strict voltage regulation is not typically necessary. Excitation frequency should be controlled within ± 1% to maintain accuracy.

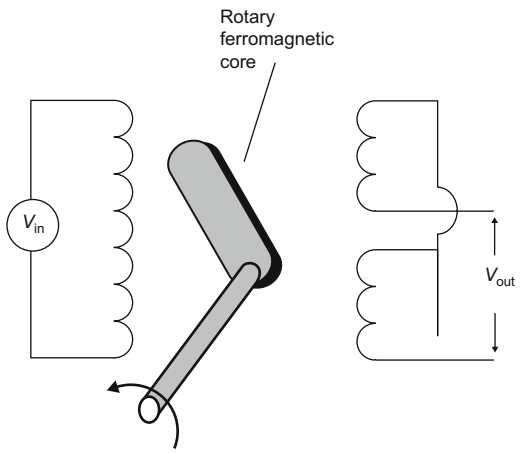


FIGURE 4.6

The working principles of a typical RVDT.

4.3 INDUSTRIAL VALVES

An industrial valve is a device that controls not only the flow of a fluid, but also the rate, volume, pressure or direction of liquids, gases, slurries, or dry materials through a pipeline, chute, or similar passageway. Using valves, the flow of a fluid in various passageways can be turned ON and OFF, regulated, modulated, or isolated. The size range available is huge from a fraction of an inch to as large as 30 ft in diameter, and they can vary in complexity from a simple brass valve available at a hardware store, to a precision-designed, highly sophisticated coolant system control valve, made of an exotic metal alloy to be used in a nuclear reactor. Industrial valves can control the flow of all types of fluid, from the thinnest gas to highly corrosive chemicals, superheated steam, abrasive slurries, toxic gases, and radioactive materials. In addition, they can handle temperatures from the cryogenic region to those of molten metal, and pressures from high vacuum to thousands of pounds per square inch.

4.3.1 Control valves

The final control element is the device that implements the control strategy determined by the output of the controller. This can be a damper or a variable speed drive pump or an ON OFF switching device, but the most commonly occuring is the control valve. It is the kind of final element that manipulates a flowing fluid, such as gas, steam, water, or chemical compound, so as to compensate for load disturbances and keep the regulated process variable as close as possible to the desired set point.

Control valves really refer to a control valve assembly. This typically consists of the valve body, internal trim parts, an actuator to provide the motive power to operate the valve, and a variety of additional valve accessories, which can include positioners, transducers, supply pressure regulators, manual operators, snubbers, or limit switches. Figure 4.7 gives two conceptual diagrams for the linear and rotary control valve assembly, respectively. The interested reader can identify the valve body, the internal trim parts, and an actuator in this diagram.

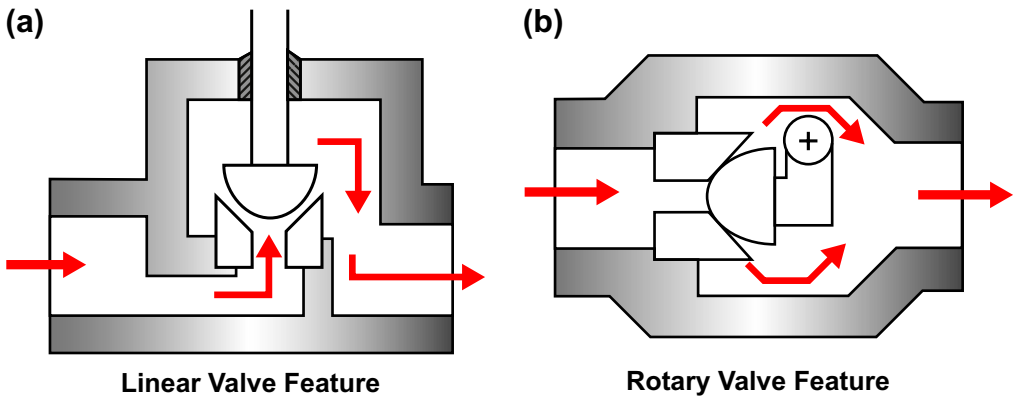


FIGURE 4.7

A conceptual diagram of control valves.

The control valve regulates the rate of fluid flow as the position of the valve plug or disk is changed by force generated by the actuator. To do this, the valve must contain fluid without external leakage; have adequate capacity for the intended service; be capable of withstanding the erosive, corrosive and temperature influences of the process; and incorporate appropriate end connections to mate with adjacent pipelines. The actuator attachment must transmit the actuator thrust to the valve plug stem or rotary shaft.

(1) Basic types

Many styles of control valve bodies have been developed through the years. Some have found wide application; others meet specific service conditions and are used less frequently. Figure 4.8 is a classification of control valves used in today's industrial controls. Figure 4.9 is a product picture of a special control valve.

The following summary describes some popular control valve body styles currently, in use some special application valves, steam conditioning valves, and ancillary devices including valve actuators, positioners, and accessories.

(1) Linear globe valves

Linear globe valves have a linear motion closure member, one or more ports, and a body distinguished by a globular-shaped cavity around the port region. Globe valves can be divided into single-ported valve bodies, balance-plug cage-guided bodies, high-capacity cage-guided valve bodies, port-guided single-port valve bodies, double-ported valve bodies, and three-way valve bodies.

(2) Rotary shaft valves

Rotary shaft valves have a flow closure member (full ball, partial ball, disk, or plug) is rotated in the flow stream to control the capacity of the valve. Rotary shaft valves can be further classified as butterfly

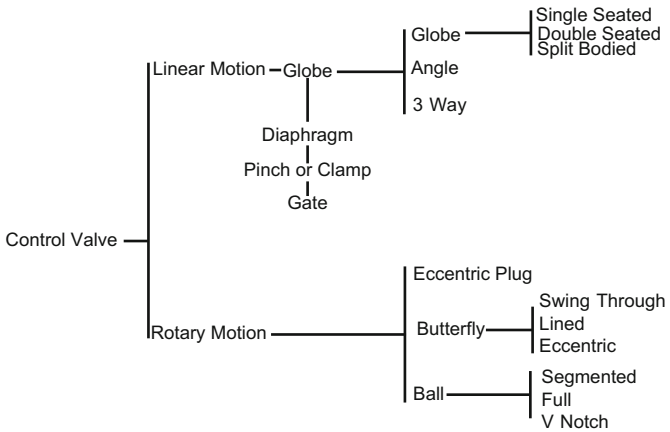


FIGURE 4.8

A classification of control valves.

FIGURE 4.9

A picture of modern control valves utilizing digital valve controllers.

(Courtesy of Emerson.)

valve bodies, V-notch ball control valve bodies, eccentric-disk control valve bodies, and eccentric-plug control valve bodies.

(3) Special valves

Standard control valves can handle a wide range of control applications, but corrosiveness and viscosity of the fluid, leakage rates, and many other factors demand consideration even for standard conditions. The following discusses some special control valve modifications that are useful in severe controlling applications.

(a) High-capacity control valves including globe style valves larger than 12-inch, ball valves over 24-inch, and high-performance butterfly valves larger than 48-inch. As valve sizes increase

arithmetically, static pressure loads at shutoff increase geometrically. Consequently, shaft strength, bearing loads, unbalance forces, and available actuator thrust all become more significant with increasing valve size. Normally, maximum allowable pressure drop is reduced on large valves to keep design and actuator requirements within reasonable limits. Even with lowered working pressure ratings, the flow capacity of some large-flow valves remains tremendous.

(b) Low-flow control valves. Many applications exist in laboratories, pilot plants and the general processing industries where control of extremely low flow rates is required. These applications are commonly handled with special trim which is often available in standard control valve bodies. This is typically made up of a seat ring and valve plug that have been designed and machined to very close tolerances, in order to allow accurate control of very small flows.

(c) High-temperature control valves. These control valves are specially designed for service at temperatures above 450°F (232°C). They are encountered in boiler feed water systems and superheater bypass systems.

(d) Cryogenic service valves. These can cope with materials and processes at temperatures below −150°F (−101°C). Many of the same issues need consideration for cryogenic conditions as with high-temperature control valves. Packing is a concern in cryogenic applications, since plastic and electrometric components often cease to function appropriately at temperatures below 0°F (−18°C). For plug seals, a standard soft seal will become very hard and less pliable, thus not providing the shut-off required. Special electrometrics have been applied in these temperatures, but require special loading to achieve a tight seal.

(4) Steam conditioning valves

A steam conditioning valve is used for the simultaneous reduction of steam pressure and temperature to the level required for a given application. Frequently, these applications deal with high inlet pressures and temperatures and require significant reductions of both properties. They have forged and fabricated bodies that can better withstand steam loads at elevated pressures and temperatures. Forged materials can withstand higher design stresses, have improved grain structure, and an inherent material integrity over cast valve bodies. The forged construction also allows the manufacturer to provide up to Class 4500, as well as intermediate and special class ratings, with greater ease vs. cast valve bodies.

Due to extreme and frequent changes in steam properties as a result of the temperature and pressure reduction, the forged and fabricated valve body design allows for the addition of an expanded outlet to control outlet steam velocity at lower pressure. Similarly, with reduced outlet pressure, the forged and fabricated design allows the manufacturer to provide different pressure class ratings for the inlet and outlet connections to more closely match the adjacent piping.

The latest versions of steam conditioning valves have feed-forward, manifold, pressure reduction-only, or turbine bypass designs.

The turbine bypass system has evolved over the past few decades as the mode of power plant operations has changed. It is employed routinely in utility power plants, where quick response to wide swings in energy demand is needed. A typical power plant operation might start at minimum load, increase to full capacity for most of the day, rapidly reduce back to minimum output, and then up again to full load all within a 24-hour period. Boilers, turbines, condensers, and other associated equipment cannot respond properly to such rapid changes without some form of turbine bypass system. This system the boiler to operate independently of the turbine. In the start-up mode, or for a rapid reduction

in demand, the turbine bypass not only supplies an alternative flow path for the steam, but conditions the steam to the same pressure and temperature that is normally produced by the turbine expansion process. By providing an alternative flow path for the steam, the turbine bypass system protects the turbine, boiler, and condenser from damage that may occur from thermal and pressure excursions. For this reason, many such systems require extremely rapid open/close response times for maximum equipment protection. This is accomplished by using an electrohydraulic actuation system, which provides both the forces and controls for such operation. Additionally, when commissioning a new plant, the turbine bypass system allows start-up and check-out of the boiler separately from the turbine. This allows quicker plant start-ups, which results in attractive economic gains. It also means that this closed-loop system can prevent atmospheric loss of treated feed water and reduction of ambient noise emissions.

(2) Valve actuators

Pneumatically operated control valve actuators are the most popular type in use, but electric, hydraulic, magnetic, and piezoelectric actuators are also popular. All these actuators were introduced in the previous chapter.

The spring-and-diaphragm pneumatic actuator is most commonly specified due to its dependability and simplicity of design. Pneumatically operated piston actuators provide high stem force output for demanding service conditions. Adaptations of both spring-and-diaphragm and pneumatic piston actuators are available for direct installation on rotary shaft control valves. Electric and electrohydraulic actuators are more complex and more expensive than pneumatic actuators, which offer advantages where no air supply source is available, where low ambient temperatures could freeze condensed water in pneumatic supply lines, or where unusually large stem forces are needed.

Pneumatically operated diaphragm actuators use the air supply from a controller, positioner, or other source. There are various styles; the direct action of increasing air pressure pushes down the diaphragm and extends the actuator stem; the reverse action of increasing air pressure pushes up the diaphragm and retracts the actuator stem; reversible actuators that can be assembled for either direct or reverse action; direct-acting unit for rotary valves where increasing air pressure pushes down on the diaphragm, which may either open or close the valve, depending on the orientation of the actuator lever on the valve shaft.

Piston actuators are pneumatically operated using high-pressure plant air to 150 psig, often eliminating the need for a supply pressure regulator. Piston actuators furnish maximum thrust output and fast stroking speeds. Piston actuators are double acting to give maximum force in either direction, or spring return to provide fail-open or fail-closed operation.

Electrohydraulic actuators require only electrical power to the motor and an electrical input signal from the controller. Electrohydraulic actuators are ideal for isolated locations where pneumatic supply pressure is not available, but where precise control of valve plug position is needed. Units are normally reversible by making minor adjustments and can be self-contained, including motor, pump, and double-acting hydraulically operated piston within a weatherproof or explosion-proof casing.

Rack and pinion designs provide a compact and economical solution for rotary shaft valves. Because of backlash, they are typically used for ON OFF applications, or where process variability is not a concern.

Traditional electric actuator designs use an electric motor and some form of gear reduction to move the valve. Through adaptation, these mechanisms have been used for continuous control with varying degrees of success. To date, electric actuators have been much more expensive than pneumatic ones for the same performance levels. This is an area of rapid technological change, and future designs may cause a shift toward greater use of electric actuators.

(3) Valve positioners

Positioners are used for pneumatically operated valves that depend on a positioner to take an input signal from a process controller and convert it into valve travel. These positioners are mostly available in three configurations:

1. Pneumatic. A pneumatic signal (usually 3 15 psig) is supplied to the positioner. The positioner translates this to a required valve position and supplies the valve actuator with the air pressure required to move the valve to the correct position.
2. Analog I/P. This positioner performs the same function as the one above, but uses electrical current (usually 4 20 mA) instead of air as the input signal.
3. Digital. Although this positioner functions very much like the analog I/P described above, it differs in that the electronic signal conversion is digital rather than analog. The digital products cover three categories:
   (a) Digital noncommunicating. A current signal (4 20 mA) is supplied to the positioner, which both powers the electronics and controls the output.
   (b) HART. This is the same as the digital noncommunicating, but is also capable of two-way digital communication over the same wires that are used for the analog signal.
   (c) Fieldbus. This type receives digitally based signals and positions the valve using digital electronic circuitry coupled to mechanical components.

(4) Valve accessories

Valve accessories include the following devices:

1. Limit switches. Limit switches operate discrete inputs to a distributed control system, signal lights, small solenoid valves, electric relays, or alarms. An assembly that can be mounted on the side of the actuator houses the switches. Each switch adjusts individually, and can be supplied for either alternating current or direct current systems. Other styles of valve-mounted limit switches are also available.
2. Solenoid valve manifold. The actuator type and the desired fail-safe operation determine the selection of the proper solenoid valve. They can be used on double-acting pistons or single-acting diaphragm actuators.
3. Supply pressure regulator. Supply pressure regulators, commonly called airsets, reduce plant air supply to valve positioners and other control equipment. Common reduced air supply pressures are 20, 35, and 60 psig. The regulator mounts integrally to the positioner or nipple-mounts or bolts to the actuator.
4. Pneumatic lock-up systems. Pneumatic lock-up systems are used with control valves to lock in existing actuator loading pressure in the event of supply pressure failure. These devices can be used with volume tanks to move the valve to the fully open or closed position on loss of

pneumatic air supply. Normal operation resumes automatically with restored supply pressure. Functionally similar arrangements are available for control valves using diaphragm actuators.

5. Fail-safe systems for piston actuators. In these fail-safe systems, the actuator piston moves to the top or bottom of the cylinder when supply pressure falls below a predetermined value. The volume tank, charged with supply pressure, provides loading pressure for the actuator piston when supply pressure fails, thus moving the piston to the desired position. Automatic operation resumes, and the volume tank is recharged when supply pressure is restored to normal.
6. PC diagnostic software. PC diagnostic software provides a consistent, easy to use interface to every field instrument within a plant. For the first time, a single resource can be used to communicate and analyze field electronic "smart" devices such as pressure xmtrs, flow xmtrs, etc., not pneumatic positioners, boosters. Users can benefit from reduced training requirements and reduced software expense. A single purchase provides the configuration environment for all products. Products and services are available that were not possible with stand-alone applications. The integrated product suite makes higher-level applications and services possible.
7. Electropneumatic transducers. The transducer receives a direct current input signal and uses a torque motor, nozzle flapper, and pneumatic relay to convert the electric signal to a proportional pneumatic output signal. Nozzle pressure operates the relay and is piped to the torque motor feedback bellows to provide a comparison between input signal and nozzle pressure.

4.3.2 Self-actuated valves

Self-actuated valves are those valves that use fluid or gas existing in a system to position the valve. Check valves and relief valves are two important examples of self-actuated valves. In addition, safety valves and steam traps are also defined as self-actuated valves. All of these valves are actuated with the system fluid or gas; no source of power outside the system fluid or gas energy is necessary for their operation.

(1) Check valves

Check valves are self-activating safety valves that permit gases and liquids to flow in only one direction, thus preventing process flow from reversing. When open and under flow pressure, the checking mechanism will move freely in the medium, offering very little resistance and minimal pressure drop. Check valves are classified as one-way directional valves; fluid flow in the desired direction opens the valve, while backflow forces the valve to close.

A check valve is hence a uni-directional valve for fluid flow. There are many ways to achieve such flow. Most check valves contain a ball that sits freely above the seat, which has only one through-hole. The ball has a slightly larger diameter than that of the through-hole. When the pressure behind the seat exceeds that above the ball, liquid flows through the valve; however, once the pressure above the ball exceeds the pressure below the seat, the ball returns to rest in the seat, forming a seal that prevents backflow.

Figure 4.10 is used here to describe briefly the principles of check valves. Device A (Figure 4.10(A)) consists of a ball bearing retained by a spring. Fluid flowing to the right will push the ball bearing against the spring, and open the valve to permit flow. This device requires some pressure to compress the spring and open the valve. If an attempt is made to flow fluid to the left, the ball bearing seals against the opening and no flow is allowed. This is a modern design that requires round balls. Device B (Figure 4.10(B)) is

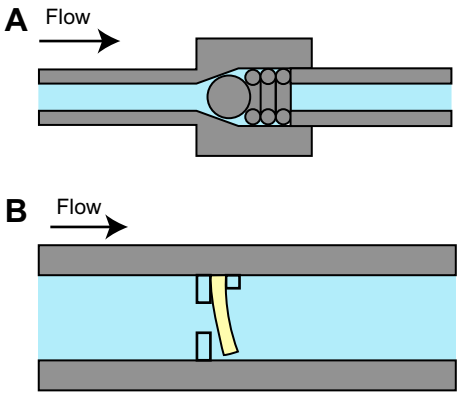


FIGURE 4.10

Device A consists of a ball bearing retained by a spring; device B is simply a flapper anchored on one side.
(Courtesy of Michigan State University.)

simply a flapper that is anchored on one side. The flapper can be a hinged metallic door, a thin piece of metal, or a piece of rubber or polymer. This is the simplest design and was used in early pumps.

Two methods of incorporating check valves into pumps are also shown in Figure 4.11. In schematic A (Figure 4.11(A)), the check valves permit expulsion of the fluid to the right on the downward stroke

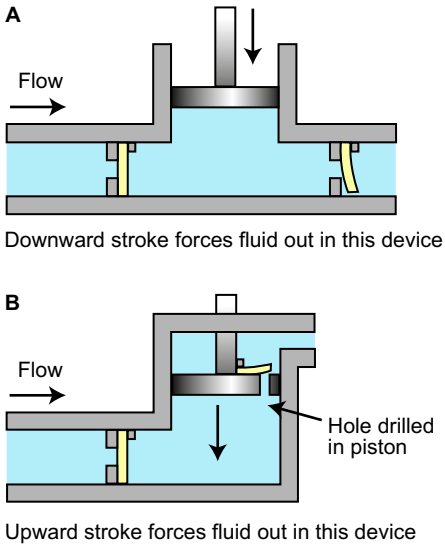


FIGURE 4.11

Two methods of incorporating check valves into pumps.
(Courtesy of Michigan State University.)

while denying flow to the left. On the upward stroke, the pump fills from the left, while denying reverse flow from the right. The design in schematic B (Figure 4.11(B)) is somewhat different. The piston has one or more holes drilled though it, with a check valve on each hole. (One hole is illustrated.) On the downward stroke, the fluid moves from below the piston, to the chamber above the piston, and is denied exit to the left. On the upward stroke, fluid is pushed out the exit on the right and simultaneously more fluid is drawn from the entrance on the left. Case B is the design illustrated by Watt in his patent and described as the air pump, since it pumps air as well as water.

Check valves use a variety of technologies in order to allow and prevent the flow of liquids and gases. Single-disk swing valves are designed with the closure element attached to the top of the cap. Double-disk or wafer check valves consist of two half-circle disks that are hinged together, and fold together upon positive flow and retract to a full circle to close against reverse flow. Lift-check valves feature a guided disk. Spring-loaded devices can be mounted vertically or horizontally. Silent or center guide valves are similar to lift check valves, with a center guide extending from inlet to outlet ports. The valve stopper is spring and bushing actuated to keep the movement "quiet". Ball check valves use a free-floating or spring-loaded ball resting in a seat ring as the closure element. Cone check valves use a free-floating or spring-loaded cone resting in the seat ring as the closure element. Although there are many types of check valves, two basic types are most popular in industrial control swing check valves and ball check valves. Both types of valves may be installed vertically or horizontally.

(1) Swing check valves

Swing check valves are used to prevent flow reversal in horizontal or vertical pipelines (vertical pipes or pipes in any angle from horizontal to vertical with upward flow only). Swing check valves have disks that swing open and closed. They are typically designed to close under their own weight, and may be in a state of constant movement if the pressure due to flow velocity is not sufficient to hold it valve in a wide-open position. Premature wear or noisy operation of the swing check valves can be avoided by selecting the correct size on the basis of flow conditions.

The minimum velocity required to hold a swing check valve in an open position is expressed by the empirical formula given by Figure 4.12, where V is the liquid flow measured in m/s or ft/s; v is the special volume of the liquid measured in m^3/N or ft^3/lb; j equals 133.7 (35) for Y-pattern, or 229.1 (60) for bolted cap, or 381.9 (100) for UL listed.

Tilting disk check valves are pivoted circular disks mounted in a cylindrical housing. These check valves have the ability to close rapidly, thereby minimizing slamming and vibrations. Tilting disk checks are used to prevent reversal in horizontal or vertical-up lines similar to swing check valves. The minimum velocity required for holding a tilting check valve wide open can be determined by the empirical formula given in Figure 4.12, where V is the liquid flow measured in m/s or ft/s; v is the special volume of the liquid measured in m^3/N or ft^3/lb; j equals 305.5 (80) for 5 times by disk angle (typically for steel), or = 114.6 (30) for 15 times by disk angle (typical for iron).

$$V \bullet j\sqrt{v}$$

FIGURE 4.12

The minimum velocity formula of swing check valves.

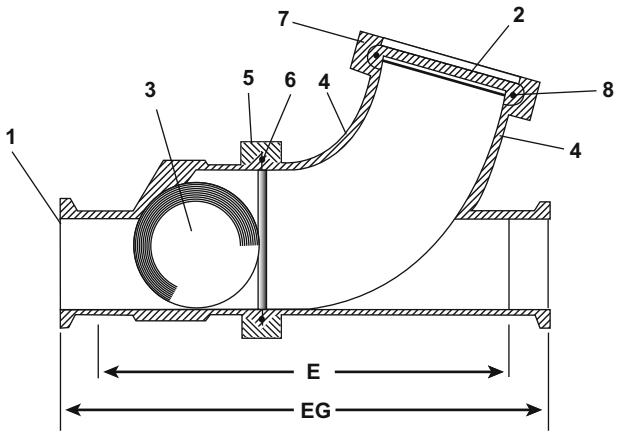


FIGURE 4.13

The working block of a ball check valve. In this figure, 1 is the seat body, 2 is the cap, 3 is the ball, 4 is the angle body, 5 is the body clamp, 6 is the body gasket, 7 is the cap clamp, 8 is the cap gasket.

(Courtesy of VNE Corporation.)

Lift check valves also operate automatically by line pressure. They are installed with pressure under the disk. A lift check valve typically has a disk that is free-floating and is lifted by the flow. Liquid has an indirect line of flow, so the lift check restricts the flow. Because of this, lift check valves are similar to globe valves, and are generally used as companions to them.

(2) Ball check valves

A ball check valve is a type of check valve in which the movable part is a spherical ball, as illustrated in Figure 4.13. Ball check valves are used in spray devices, dispenser spigots, manual and other pumps, and refillable dispensing syringes. Ball check valves may sometimes use a free-floating or spring-loaded ball.

Ball check valves generally have simple, inexpensive metallic parts, although specialized ball check valves are also available. For example, ball check valves in high-pressure pumps used in analytical chemistry have a ball of synthetic ruby, which is a hard and chemically resistant substance. A ball check valve is not to be confused with a ball valve, which is a quarter-turn valve similar to a butterfly valve in which the ball acts as a controllable rotor.

(2) Relief valves

A relief valve is a mechanism that ensures system fluid flow when a preselected differential pressure across the filter element is exceeded; the valve allows all or part of the flow to bypass the filter element. Relief valves are used in oil and gas production systems, compressor stations, gas transmission (pipeline) facilities, storage systems, in all gas processing plants, and whenever there is a need to exhaust the overpressure volume of gas, vapor, and/or liquid.

Figure 4.14 illustrates the working blocks of a relief valve, which operate on the principle of unequal areas being exposed to the same pressure. When the valve is closed, the system pressure

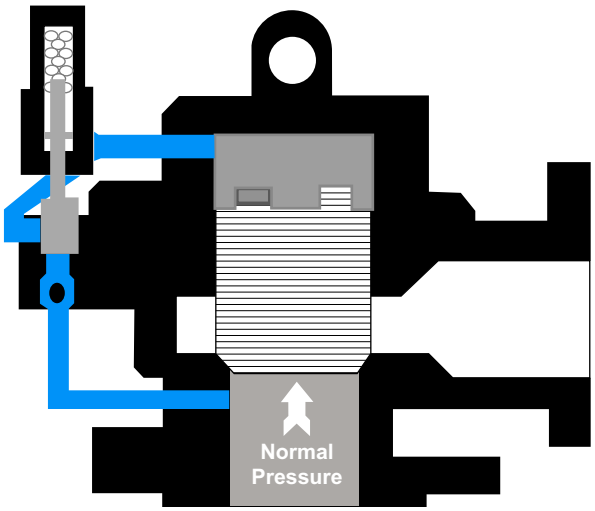


FIGURE 4.14

The working blocks of relief valve.

(Courtesy of P.C. McKenzie Company)

pushes upward against the piston seat seal on an area equal to the inside diameter of the seat. Simultaneously, the same system pressure, passing through the pilot, exerts a downward force on the piston, but this acts on an area approximately 50% greater than the inside diameter of the seat. The resulting differential force holds the valve tightly closed. As the system reaches the discharge pressure set for the valve, the piston seal becomes tighter, until the system pressure reaches the relief valve discharge set pressure. At that moment, and not before, the pilot cuts off the supply of system pressure to the top of the piston and vents that system pressure which is located in the chamber above the piston of the relief valve. At the same instant, the relief valve pops open. When the predetermined blow-down pressure is reached (either fixed or adjustable), the pilot shuts off the exhaust and reopens flow of system pressure to the top of the piston, effectively closing the relief valve.

Figure 4.15 is a drawing of a direct-operating pressure relief valve. This pressure relief valve is mounted at the pressure side of the hydraulic pump that is located on its bottom. The task of this pressure relief valve is to limit the pressure in the system to an acceptable value. A pressure relief valve has infact the same construction as a spring-operated check valve. When the system becomes overloaded, the pressure relief valve will open, and the pump flow will be led directly into the hydraulic reservoir. The pressure in the system remains at the value determined by the spring on the pressure relief valve. In the pressure relief valve, the pressure that is equal to the system energy will be converted into heat. For this reason pressure relief valves should not be operated for long durations.

Table 4.2 lists several important types of relief valve, categorized in terms of their applications. In industrial control, typical features for relief valves include the following:

1. Pressure settings of relief valves are externally adjustable while the valve is in operation. Most vendors of relief valves offer eight different spring ranges, to provide greater system sensitivity and enhanced performance.

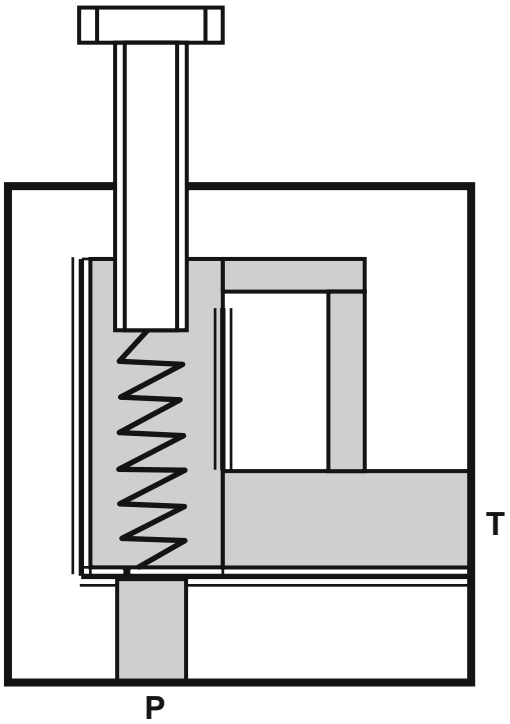


FIGURE 4.15

A drawing of a direct-operating pressure relief valve.

Table 4.2 Important Types of Relief Valves

Temperature and pressure relief valves	Temperature and pressure relief valves are used in water heater and hot water storage tank applications to provide automatic temperature and pressure protection to hot water supply tanks and hot water heaters.
Reseating temperature and pressure relief valves	Automatic reseating temperature and pressure relief valves are used in commercial water heater applications to provide automatic temperature and pressure protection to domestic hot water supply tanks and hot water heaters.
Pressure relief valves	Pressure relief valves are used in hot water heating and domestic supply boiler applications to protect against excessive pressures on all types of hot water heating supply boiler equipment.
Poppet-style relief valves	Calibrated pressure relief valves are used in commercial, residential and industrial applications to protect against excessive pressure in systems containing water, oil, or air.

2. Manual override option with positive stem retraction is available for pressures. This option permits the user to relieve upstream pressure while maintaining the predetermined cracking pressure.
3. Color-coded springs and labels indicate spring cracking range.
4. Lock-wire feature secures a given pressure setting.

4.3.3 Solenoid valves

A solenoid control valve is a kind of isolation valve that is an electromechanical device allowing an electrical device to control the flow of gas or liquid. The electrical device causes a current to flow through a coil located on the solenoid valve, which in turn results in a magnetic field that causes the displacement of a metal actuator. The actuator is mechanically linked to a valve inside the solenoid valve. This mechanical valve then opens or closes and so allows a liquid or gas either to flow through, or be blocked by the solenoid valve. In this control system, a spring is used to return the actuator and valve back to their resting states when the current flow is removed. Figure 4.16 shows the application of a solenoid valve in a typical control system. A coil inside the solenoid valve generates a magnetic field once an electric current is flowing through. The generated magnetic field actuates the ball valve that can change states to open or close the device in the fluid direction indicated by the arrow.

Solenoid valves are used wherever fluid flow has to be controlled automatically, such as in factory automation. A computer running an automation program to fill a container with some liquid can send

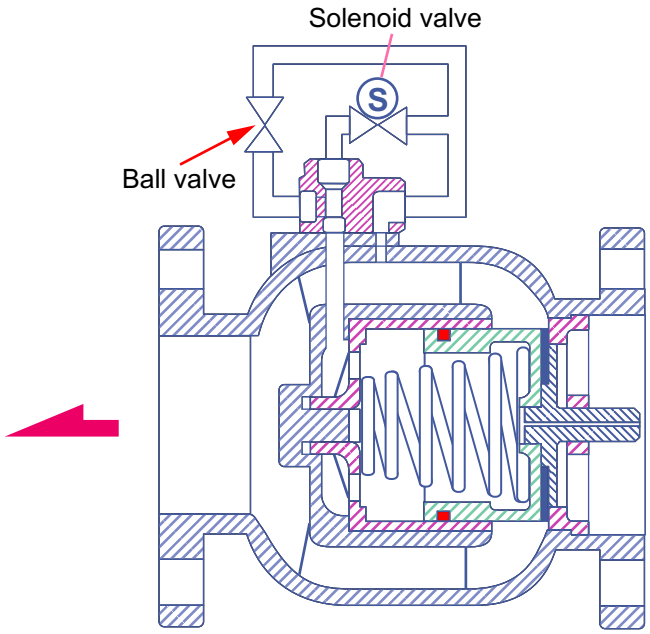


FIGURE 4.16

A typical flow control system with solenoid valve.

(Courtesy of Z Tide Valves.)

a signal to the solenoid valve to open, allowing the container to fill, and then remove the signal to close the solenoid valve, and so stop the flow of liquid until the next container is in place. A gripper for grasping items on a robot is frequently an air-controlled device. A solenoid valve can be used to allow air pressure to close the gripper, and a second solenoid valve can be used to open the gripper. If a two-way solenoid valve is used, two separate valves are not needed in this application. Solenoid valve connectors are used to connect solenoid valves and pressure switches.

(1) Operating principles

Solenoid valves are control units which, when electrically energized or de-energized, either shut off or allow fluid flow. The actuator inside a solenoid valve takes the form of an electromagnet. When energized, a magnetic field builds up, which pulls a plunger or pivoted armature against the action of a spring. When de-energized, the plunger or pivoted armature is returned to its original position by the spring action.

According to the mode of actuation, a distinction is made between direct-acting valves, internally piloted valves, and externally piloted valves. A further distinguishing feature is the number of port connections or the number of flow paths or "ways".

Direct-acting solenoid valves have the seat seal attached to the solenoid core. In the de-energized state, the seat orifice is closed, and opens when the valve is energized. With direct-acting valves, the static pressure forces increase with increasing orifice diameter, which means that the magnetic forces required for overcoming the pressure force become correspondingly larger. Internally piloted solenoid valves are therefore employed for switching higher pressures in conjunction with larger orifice sizes; in this case, the differential fluid pressure performs most of the work of opening and closing the valve.

Two-way solenoid valves are shut-off valves with one inlet port and one outlet port, as shown in Figure 4.17(a). In the de-energized state, the core spring, assisted by the fluid pressure, holds the valve seal down on the valve seat to shut off the flow. When energized, the core and seal are pulled into the solenoid coil and the valve opens. The electromagnetic force is greater than the combined spring force and the static and dynamic pressure forces of the medium.

Three-way solenoid valves have three port connections and two valve seats. One valve seal always remains open and the other closed in the de-energized mode. When the coil is energized, the mode reverses. The three-way solenoid valve shown in Figure 4.17(b) is designed with a plunger-type core. Various valve operations are available, according to how the fluid medium is connected to the working ports. In Figure 4.17(b), The fluid pressure builds up under the valve seat. With the coil is de-energized, a conical spring holds the lower core seal tightly against the valve seat and shuts off the fluid flow. Port A is exhausted through R. When the coil is energized, the core is pulled in, and the valve seat at Port R is sealed off by the spring-loaded upper core seal. The fluid medium now flows from P to A.

Unlike the versions with plunger-type cores, pivoted-armature solenoid valves have all port connections inside the valve body. An isolating diaphragm ensures that the fluid medium does not come into contact with the coil chamber. Pivoted-armature valves can be used to obtain any three-way solenoid valve operation. The basic design principle is shown in Figure 4.17(c). Pivoted-armature valves are provided with manual override as a standard feature.

Internally piloted solenoid valves are fitted with either a two-way or a three-way pilot solenoid valve. A diaphragm or a piston provides the seal for the main valve seat. The operation of such a valve is indicated in Figure 4.17(d). When the pilot valve is closed, the fluid pressure builds up on both sides of the diaphragm via a bleed orifice. As long as there is a pressure differential between the inlet and

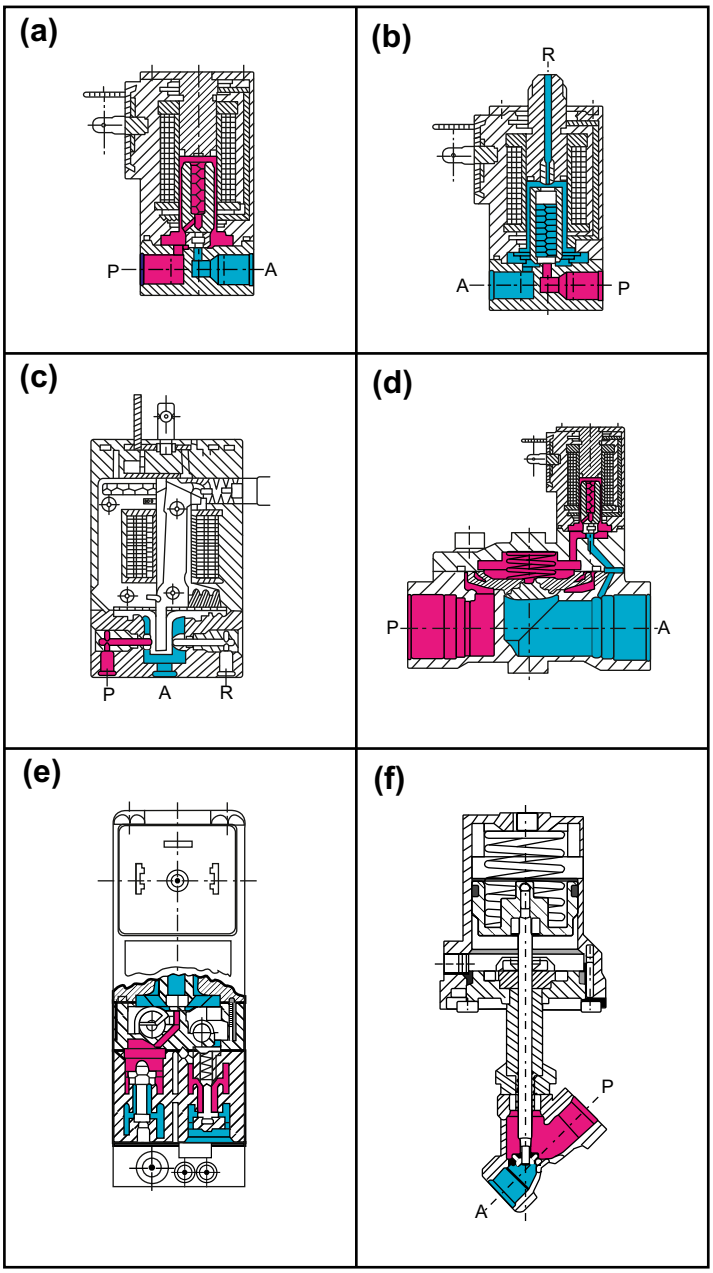


FIGURE 4.17

The operating principle of solenoid valves.

(Courtesy of OMEGA)

outlet ports, a shut-off force is available by virtue of the larger effective area on the top of the diaphragm. When the pilot valve is opened, the pressure is relieved from the upper side of the diaphragm. The greater effective net pressure force from below now raises the diaphragm and opens the valve. In general, internally piloted valves require a minimum pressure differential to ensure satisfactory opening and closing.

Internally piloted four-way solenoid valves are used mainly in hydraulic and pneumatic applications to actuate double-acting cylinders. These valves have four port connections; a pressure inlet P, two cylinder port connections A and B, and one exhaust port connection R. An internally piloted four/two-way poppet solenoid valve is shown in Figure 4.17(e). When de-energized, the pilot valve opens at the connection from the pressure inlet to the pilot channel. Both poppets in the main valve are now pressurized and switch over. Now port connection P is connected to A, and B can exhaust via a second restrictor through R.

With these types an independent pilot medium is used to actuate the valve. Figure 4.17(f) shows a piston-operated angle-seat valve with closure spring. In the unpressurized condition, the valve seat is closed. A three-way solenoid valve, which can be mounted on the actuator, controls the independent pilot medium. When the solenoid valve is energized, the piston is raised against the action of the spring and the valve opens. A normally open valve version can be obtained if the spring is placed on the opposite side of the actuator piston. In these cases, the independent pilot medium is connected to the top of the actuator. Double-acting versions controlled by four/two-way valves do not contain any spring.

(2) Basic types

Solenoid valves are opened and closed via a solenoid that is activated by an electrical signal. In most industrial applications, solenoid valves are of the following five types.

(1) Two-way solenoid valves

This type of solenoid valve normally has one inlet and one outlet, and is used to permit and shut off fluid flow. The two types of operations for this type are “normally closed” and “normally open”.

(2) Three-way solenoid valves

These valves normally have three pipe connections and two orifices. When one orifice is open, the other is closed and vice versa. They are commonly used to alternately apply pressure to an exhaust pressure from a valve actuator or a single-acting cylinder. These valves can be normally closed, normally open, or universal.

(3) Four-way solenoid valves

These valves have four or five pipe connections, commonly called ports. One is a pressure inlet port, and two others are cylinder ports providing pressure to the double-acting cylinder or actuator, and one or two outlets exhaust pressure from the cylinders. They have three types of construction; single solenoid, dual solenoid, or single air operator.

(4) Direct-mount solenoid valves

These are two-way, three-way, and four-way solenoid valves that are designed for gang mounting into different quantities of valves. Any combination of normally closed, normally open, or universal valves

may be grouped together. These series are standard solenoid valves whose pipe connections and mounting configurations have been replaced by a mounting configuration that allows each valve to be mounted directly to an actuator without the use of hard piping or tubing.

(5) Manifold valves

A manifold of solenoid valves consists of a matrix of solenoid valves mounted in modules on a skid with adjustable legs along one direction (Figure 4.18). The number of valves depends on the elements to be connected and on the functions of each of these elements. A plurality of solenoid valves is arranged and placed on the installation face of the manifold, and a board formed with an electric circuit for feeding these solenoid valves (Figure 4.18). Each solenoid valve includes a valve portion containing a valve member and a solenoid operating portion for driving the valve member. The board is mounted on the first side face of the manifold under the solenoid operating portion. The board can be attached and detached while leaving the solenoid valves mounted on the manifold, feeding connectors and indicating lights being provided in positions on the board corresponding to the respective solenoid valves. Each feeding connector is disposed in such a position that it is connected to a receiving terminal of the solenoid valve in a plug-in manner when mounting the solenoid valve on the manifold. Each indicating light is disposed in such a position that it can be visually recognized from above the solenoid valve while leaving the solenoid valve mounted on the manifold.

This manifold allows the centralizing of the functions of one or various tanks in a modular way, enhancing the efficiency of the system and the degree of control over the process. The solenoid valve manifold is an automated alternative to flexible hoses and flow divert panels with changeover bends. As many valves as the number of functions the element has to perform are connected to the tank or working line. No manual operation is required. The operation is automated, preventing any risk of accidents.

FIGURE 4.18

Several types of manifold of solenoid valves.

(Courtesy of KIP Inc.)

4.3.4 Float valves

Float control systems that monitor liquid or powder levels in containers such as tanks and reservoirs have two kinds of sensors; float switches and float valves. In a float control system, float switches are used to detect liquid or powder levels, or interfaces between liquids; float valves control the liquid or powder level in elevated tanks, standpipes, or storage reservoirs and modulate the reservoir flow to maintain a constant level.

Figure 4.19 is a float control system monitoring the liquid level of a high-temperature or high-pressure tank using both float switches and float valves. Three switches are located along the right side of this tank to detect three different levels of liquid by turning on either alarms or signals once the liquid reaches them. The two valves set at the top and bottom of this tank are used to input or output the liquid and maintain the appropriate levels in this tank.

Float control systems require both float switches and float valves to work. Figure 4.20(a) is a diagram of a sample float switch, and Figure 4.20(b) is a diagram of a sample float valve.

(1) Float switch

Float switches (Figure 4.20(a)) can be used either as alarm devices or as control switches, turning something on or off, such as a pump, or sending a signal to a valve actuator. What makes level switches special is that they have a switched output, and can be either electromechanical or solid state, either normally open or normally closed.

Float switches provide industrial control for motors that pump liquids from a sump or into a tank. For tank operation, a float operator assembly is attached to the float switch by a rod, chain, or cable. The float switch is actuated on the basis of the location of the float in the liquid. The float switch contacts are open when the float forces the operating lever to the up position. As the liquid level falls, the float and operating lever move downward. The contacts can directly activate a motor, or provide

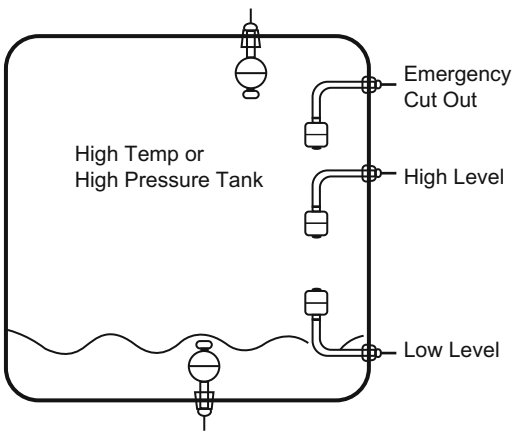


FIGURE 4.19

A schematic of a float control system for a tank installed with three float switches along the right side and two float valves at the top and the bottom.

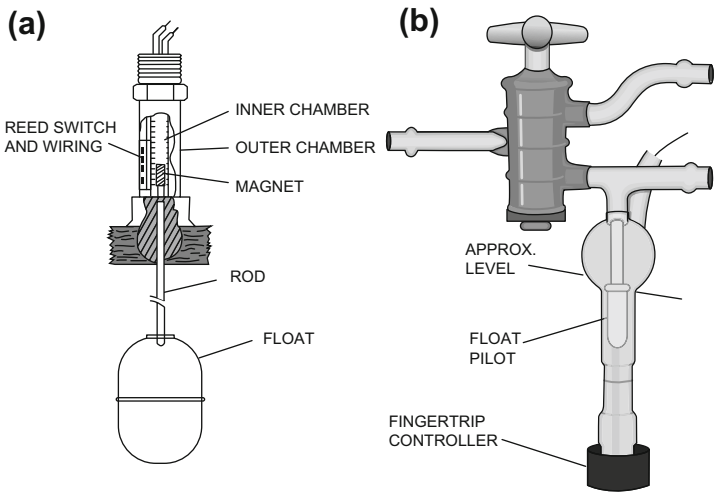


FIGURE 4.20

(a) A float switch, and (b) a float valve.

input for a logic system that can fill the tank. As the liquid level rises, the float and operating lever move upward. When the float reaches a preset high level, the float switch contacts open, deactivating the circuit and stopping the motor. Sump operation is exactly the opposite of tank operation.

(2) Float valve

A float valve (Figure 4.20(b)) is mounted on the tank or reservoir inlet, below or above the requested water level. The float pilot can either be assembled on the main valve (for above-level installation) or be connected to the main valve by a command tube. The valve closes when the water level rises, by filter discharge pressure acting with the tension spring on the top of the diaphragm in the valve cover chamber, thus raising the float to its closed position. It opens when the float descends, due to a drop in water level by filter discharge pressure acting under the valve disk. The difference between maximum and minimum levels is very small and is affected by the length of the float pilot arm. In practice, the water level is maintained continuously at the maximum point as long as the upstream flow exceeds the downstream flow.

4.3.5 Flow valves

Measuring the flow of liquids and gases is a critical need in many industrial processes, such as chemical packaging, wastewater treatments, and aircraft manufacturing. In some industrial operations, the ability to conduct accurate flow measurements is so important that it can make the difference between making a profit and taking a loss. In other cases, a failure in flow control can cause serious (or even disastrous) results. Flow control systems are designed specifically for operation with a wide range of liquids and gases in both safe and hazardous areas, in hygienic, high-temperature, and high-pressure environments, and for use with aggressive media. Some examples of such applications are pump protection, cooling circuit protection, high and low flow rate alarm, and general flow monitoring.

The technologies for gas flow meters and liquid flow meters vary widely. The most common types of operation principles are inferential flow measurement, positive displacement measurement, velocity measurement, true mass flow measurement, and thermodynamic loss measurement. The main types of operating principles for gas flow meters and liquid flow meters are given below.

(1) Gas flow switches and liquid flow switches, velocity

Gas flow switches and liquid flow switches are used to measure the flow or quantity of a moving fluid in terms of velocity. The most common types are inferential flow measurement, positive displacement, velocity meters, and true mass flow meters.

Inferential measurement refers to the indirect measurement of flow by directly measuring another parameter and inferring the flow based on well-known relationships between this parameter and flow. The use of differential pressure as an inferred measurement of a liquid's rate of flow is the most common type that is in use today.

Positive displacement meters take direct measurements of liquid flows. These devices divide the fluid into specific increments and move it on. The total flow is an accumulation of the measured increments, which can be counted by mechanical or electronic techniques. They are often used for high-viscosity fluids.

Velocity-type gas flow switches and liquid flow switches are devices that operate linearly with respect to volume flow rate. Because there is no square-root relationship, as with differential pressure devices, their range is greater.

(2) Liquid flow switches and gas flow switches, mass

Liquid and gas flow switche are devices used for measuring the flow, or quantity of a moving liquid or gas, respectively, in terms of units of mass per unit time, such as pounds per minute.

These may be sensors with electrical output or may be standalone instruments with local displays and controls. The most common types are true mass flow meters.

True mass flow meters are devices that measure mass rate of flow directly, such as thermal meters or Calorie meters. The most important parameters are the flow range to be measured, and whether liquids or gases will be the measured fluids. Also important are operating pressure, fluid temperature, required accuracy. Typical electrical outputs for mass gas flow meters and liquid flow meters are analog current, voltage, frequency, or switched output. Computer output options can be via serial or parallel interfaces. These sensors can be mounted as either inline or insertion devices. Inline sensors can be held in place by using flanges, threaded connections, or clamps. Insertion style sensors are typically threaded through a pipe wall and stick directly in the process flow.

(3) Gas flow switches and liquid flow switches, volumetric

Gas flow switches provide output based on the measured flow of a moving gas in terms of volume per unit time, such as cubic feet per minute. Liquid flow switches, volumetric, are devices with a switch output used for measuring the flow or quantity of a moving fluid in terms of a unit of volume per unit time, such as liters per minute.

A clear understanding of the requirements of the particular application is needed to select the correct type of switch. With most liquid flow measurement instruments, the flow rate is determined inferentially by measuring the liquid's velocity or the change in kinetic energy. Velocity depends on the pressure differential that is forcing the liquid through a pipe or conduit. Because the pipe's

cross-sectional area is known and remains constant, the average velocity is an indication of the flow rate. The basic relationship for determining the liquid's flow rate in such cases is $Q = V \times A$, where Q is the liquid flow through a pipe, V is the average velocity of the flow, and A is the cross-sectional area of the pipe. Other factors affecting liquid flow rate include the liquid's viscosity and density. Specific applications should be discussed with a volumetric gas flow switch manufacturer before purchasing to ensure proper fit, form, and function.

Both volumetric gas flow switches and volumetric liquid flow switches are available with four different meter types; inferential flow meters, positive displacement meters, velocity meters, and true mass flow meters. The basic operating principle of differential pressure flow meters is that the pressure drop across the meter is proportional to the square of the flow rate, so the flow rate is obtained by measuring the pressure differential calculating its and square root. Direct measurements of liquid flows can be made with positive-displacement flow meters.

(4) Pneumatic relays

Pneumatic relays control output air flow and pressure in response to a pneumatic input signal. They can perform simple functions such as boosting or scaling the output, or complex reversal, biasing, and math function operators.

Problems

1. Please explain the differences in operating principles and control applications between limit switches and position sensors, and between proximity switches and position sensors, respectively.
2. Where can limit switches be used in a car for safety controls?
3. Photoelectric switches are available with three different types of operating principle; fixed field sensing, adjustable field sensing, and background suppression through triangulation. Please analyze the physical mechanisms for these three different types of operating principles.
4. By searching the Internet, give examples for five submodes of proximity photoelectric switches; diffuse, divergent, convergent, fixed field, and adjustable field.
5. In industrial control systems, a field level switch should include the same physical sensor; for example, a limit switch should include a limit sensor; a proximity switch can include a proximity sensor. Do you accept this statement?
6. Please give an explanation of why the near field distance *N* and the pulse echo beam spread angle *a* are the most important parameters for ultrasonic transducers to work properly.
7. Think about which physical parameter for ultrasonic transducers could determine the scale of flaws detected in a testing material body.
8. To obtain accurate measurements with a dual element transducer (Figure 4.4 (a)), both of its transmitting and receiving elements should be; (1) working synchronously; (2) with a proper angle. Please explain the reasons.
9. What law of physics is the operating principle for LVDT and RVDT based on?
10. Figure 4.7 gives two conceptual diagrams for linear and rotary control valve assembly, respectively. In each of these two conceptual diagrams, please identify where the valve body, the internal trim parts, and the actuator can be positioned.
11. Please explain why all control valves must contain the fluid without external leakage.
12. By analyses of the operating principles and functions of the turbine bypass system, try to explain why the turbine bypass system is required in modern power plants.
13. Why does a self actuated valve not need an actuator?
14. A ball check valve is a type of check valve in which the movable part is a spherical ball, as illustrated in Figure 4.13. Please explain the operating process of the ball check valve in this diagram.

15. The relief valve is a valve mechanism that ensures system fluid flow when a preselected differential pressure across the filter element is exceeded; the valve allows all or part of the flow to bypass the filter element. Please analyze the relief valve mechanism establishing a preselected differential pressure in Figure 4.14.
16. Which law of physics are solenoid valves working with?
17. A gripper for grasping items on a robot is an air controlled device. A solenoid valve can be used to allow air pressure to close the gripper, and a second solenoid valve can be used to open the gripper. If a two way solenoid valve is used, two separate valves are not needed in this application. Please explain how a two way solenoid valve can replace these two separate valves on a robot in this application.

Further Reading

Honeywell (http://hpsweb.honeywell.com). http://content.honeywell.com/sensing/prodinfo/solidstate/technical/chapter2.pdf. Accessed: April 2009.

Honeywell (http://hpsweb.honeywell.com). http://www.ssec.honeywell.com/magnetic/datasheets/sae.pdf. Accessed: April 2009.

Honeywell (http://hpsweb.honeywell.com). http://sensing.honeywell.com/. Accessed: April 2009.

Honeywell (http://hpsweb.honeywell.com). http://www.honeywell.com.pl/pdf/automatyka domow/palniki do kotlow/golden/Silowniki/V4062.pdf. Accessed: April 2009.

Hydraulic Tutorial (http://www.hydraulicsupermarket.com). http://www.hydraulicsupermarket.com/technical.html. Accessed: May 2009.

Omega (http://www.omega.com). http://www.omega.com/techref/pdf/STRAIN GAGE TECHNICAL DATA.pdf . Accessed: May 2009.

Omega (http://www.omega.com). http://www.omega.com/techref/flowmetertutorial.html. Accessed: May 2009.

Omega (http://www.omega.com). http://www.omega.com/techref/techprinc.html. Accessed: May 2009.

Omron (http://www.omron.com). http://www.sti.com/switches/swdatash.htm. Accessed: May 2009.

GlobalSpec (www.globalspec.com). Types of transducers. http://search.globalspec.com/ProductFinder/FindProducts?query=types%20of%20transducer. Accessed: May 2009.

NDT (http://www.ndt ed.org). Ultrasound. http://www.ndt ed.org/EducationResources/CommunityCollege/Ultrasonics/cc ut index.htm. Accessed: May 2009.

BPO (http://www.bostonpiezooptics.com). Ultrasonic transducers. http://www.bostonpiezooptics.com/?D=15. Accessed: May 2009.

Emersion (http://www.EmersionProcess.com/Fisher). Control Valve Handbook: Fourth Edition. 2008.

PMRC (http://www.plant maintenance.com). Articles on valves and control valves. http://www.plant maintenance.com/maintenance articles valves.shtml. Accessed: May 2009.

Sony (http://www.sony.com). http://www.docs.sony.com/release/GDMC520Kguide.pdf . Accessed: April 2009.

SukHamburg (http://www.SukHamburg.de). http://www.silicon software.de/download/archive/Valves e.pdf. Accessed: April 2009.

VNE (http://www.vnestainless.com/default.aspx). Valves:http://www.vnestainless.com/default.aspx. Accessed: April 2009.

WATTS (http://www.watts.com). http://www.watts.com/pro/ products sub.asp?catId = 69 & parCat = 125. Accessed: May 2009.

Z Tide Valves (http://www.z tide.com.tw). Valves. http://www.z tide.com.tw/multi/index e.htm. Accessed: May 2009.

CHAPTER

Microprocessors

5

5.1 SINGLE-CORE MICROPROCESSOR UNITS

A microprocessor incorporates most of the entire central processing functions on a single integrated circuit (IC). The first microprocessors emerged in the early 1970s, and were used for electronic calculators. Since then, microprocessors within personal computers and industrial embedded controllers have evolved continuously, with each version being compatible with its predecessors.

The major producers of microprocessors have been Intel (Integrated Electronics Corporation) and AMD (Advanced Micro Devices, Inc.) since the early 1970s. Intel marketed the first microprocessor in 1971, named the 4004, a revolution in the electronics industry. With this processor, functionality started to be programmed by software. Am2900 is a family of integrated circuits (ICs) created in 1975 by AMD. They were constructed with bipolar devices, in a bit-slice topology, and were designed to be used as modular components, each representing a different aspect of a computer's central processing unit (CPU).

At first, these microprocessors only handled 4 bits of data at a time (a nibble), contained 2000 transistors, had 46 instructions, and allowed 4 kB of program code and 1 kB of data, and had speeds of around several MHz. However, both Intel, AMD and other corporations such as IBM and Motorola quickly expanded the microprocessors' capacities from 8, 10, 14, 16, 20, 24, 32, 64, 128 into the 256 bits of current versions; increased their transistors from 2000 into thousands of millions; and enhanced their speeds from a few MHz into a few hundred or more GHz. At the same time personal computers and industrial controllers have been developed that use these advanced microprocessors.

Microprocessors come and go, but most manufacturers know that the important differences between them are often microprocessor clock speeds and cache sizes. Although the performance of today's microprocessors continues to improve, existing architectures based on an out-of-order execution model require increasingly complex hardware mechanisms and are increasingly impeded by performance limiters such as branches and memory latency. In the mid-1980s to early-1990s, a crop of new high-performance RISC (reduced instruction set computer) microprocessors appeared with discrete RISC-like designs such as the IBM 801 and others. RISC microprocessors were initially used in special-purpose machines and UNIX workstations, but then gained wide acceptance in other roles. As of 2007, two 64-bit RISC architectures were still being produced in volume for non-embedded applications; SPARC and Power Architecture. The RISC-like Itanium is produced in smaller quantities. The vast majority of 64-bit microprocessors are now x86-64 CISC designs from AMD and Intel.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10005-1

Though the term "microprocessor" has traditionally referred to a single-core (chipset) or multicore (chipset) CPU or system-on-a-chip (SoC), several types of specialized processing devices have followed from the technology. The most common examples are microcontrollers, digital signal processors (DSP) and graphics processing units (GPU). Many of these are either not programmable, or have limited programming facilities. For example, in general, GPUs through the 1990s were mostly non-programmable and have only recently gained limited facilities such as programmable vertex shaders. There is no universal consensus on what defines a "microprocessor", but it is usually safe to assume that the term refers to a general-purpose CPU of some sort, and not a special-purpose processor unless specifically noted. The latest ones have a unique combination of innovative features, including explicit parallelism, prediction, and speculation, which are described below.

(1) Parallelism

In today's microprocessor architectures, the compiler creates sequential machine codes that attempt to mimic parallelism in hardware. The microprocessor's hardware must then reinterpret this machine code and try to identify opportunities for parallel execution; the key to faster performance. This process is inefficient not only because the hardware does not always interpret the compiler's intentions correctly, but also because it uses a valuable die area that could be better used to do real work such as executing instructions. Even today's fastest and most efficient microprocessors devote a significant percentage of hardware resources to this task of extracting more parallelism from software code.

The use of explicit parallelism enables far more effective execution of software instructions. In the newer architecture models, the compiler analyzes and explicitly identifies parallelism in the software at compile time. This allows optimal structuring of machine code to deliver the maximum performance before the processor executes it, rather than potentially wasting valuable microprocessor cycles at runtime. The result is significantly improved processor utilization, and no precious die area is wasted by the hardware reorder engine, as occurs in out-of-order reduced instruction set computer (RISC) processors.

(2) Prediction

Simple decision structures, or code branches, are a hard performance challenge to out-of-order RISC architectures. In the simple if-then-else decision code sequence, traditional architectures view the code in four basic blocks. In order to continuously feed instructions into the processor's instruction pipeline, a technique called branch prediction is commonly used to predict the correct path. With this technique, mispredicts commonly occur 5 10% of the time, causing the entire pipeline to be purged and the correct path to be reloaded. This level of misprediction can slow processing speed by as much as 30 40%.

To address this problem, and to improve performance, the new architectures use a technique known as prediction. Prediction begins by assigning special flags called predicate registers to both branch paths p1 to the "then" path and p2 to the "else" path. At run time, the compare statement stores either a true or a false value in the 1-bit predicate registers. The microprocessor then executes both paths but only the results from the path with a true predicate flag are used. Branches, and the possibility of associated mispredicts, are removed, the pipeline remains full, and performance is increased accordingly.

(3) Speculation

Memory latency is another big problem for current microprocessors' architectures. Because memory speed is significantly slower than processor speed, the microprocessor must attempt to load data from memory as early as possible to ensure that data are available when needed. Traditional architectures allow compilers and microprocessor to schedule loads before data are needed, but branches act as barriers to this load hoisting.

The new architectures employ a technique known as "speculation" to initiate loads from memory earlier in the instruction stream, even before a branch. Because a load can generate exceptions, a mechanism to ensure that exceptions are properly handled is needed to support speculation that hoists loads before branches. The memory load is scheduled speculatively above the branch in the instruction stream, so as to start memory access as early as possible. If an exception occurs, this event is stored and the "checks" instruction causes the exception to be processed. The elevation of the load allows more time to deal with memory latency, without stalling the processor pipeline. Branches occur with great frequency in most software code sequences. The unique ability of these architectures to schedule loads before branches significantly increases the number of loads that can be speculated in comparison with traditional architectures.

5.1.1 Microprocessor unit organization

The microprocessor plays a significant role in the functioning of industries everywhere. Nowadays, the microprocessor is used in a wide range of devices or systems as a digital data processing unit, or as the computing unit of an intelligent controller or a computer to control processes, or make production automated, and even to turn devices on or off. The microprocessor is a multipurpose, programmable, clock-driven, register-based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes this data according to those instructions, and provides results as output. At a very elementary level, an analogy can be drawn between microprocessor operations and the functions of the human brain, which processes information according to understandings (instructions) stored in its memory. The brain gets input from the eyes and ears and sends processed information to output "devices", such as the face with its capacity to register expression, the hands, or the feet.

A typical programmable machine can be represented by five components; microprocessor, memory, input, output, and bus. These five components work collaboratively and interactively with each other to perform a given task; thus they comprise a system. The physical components are called hardware. A set of instructions written for the microprocessor to perform a task is called a program, and a group of programs is called software. Assuming that a program and data are already entered in the memory, the microprocessor executes a program by reading all the data including instructions from the memory via the bus, and then processes these data in terms of the instructions, then writes the result into memory via the bus.

(1) Block diagram of a microprocessor unit

Figure 5.1 depicts a function block diagram of the Intel-486 GX processor chipset, which gives the microarchitecture of this Intel processor. Figure 5.2 is the block diagram for the microarchitecture of the Intel Pentium-4 processor chipset.

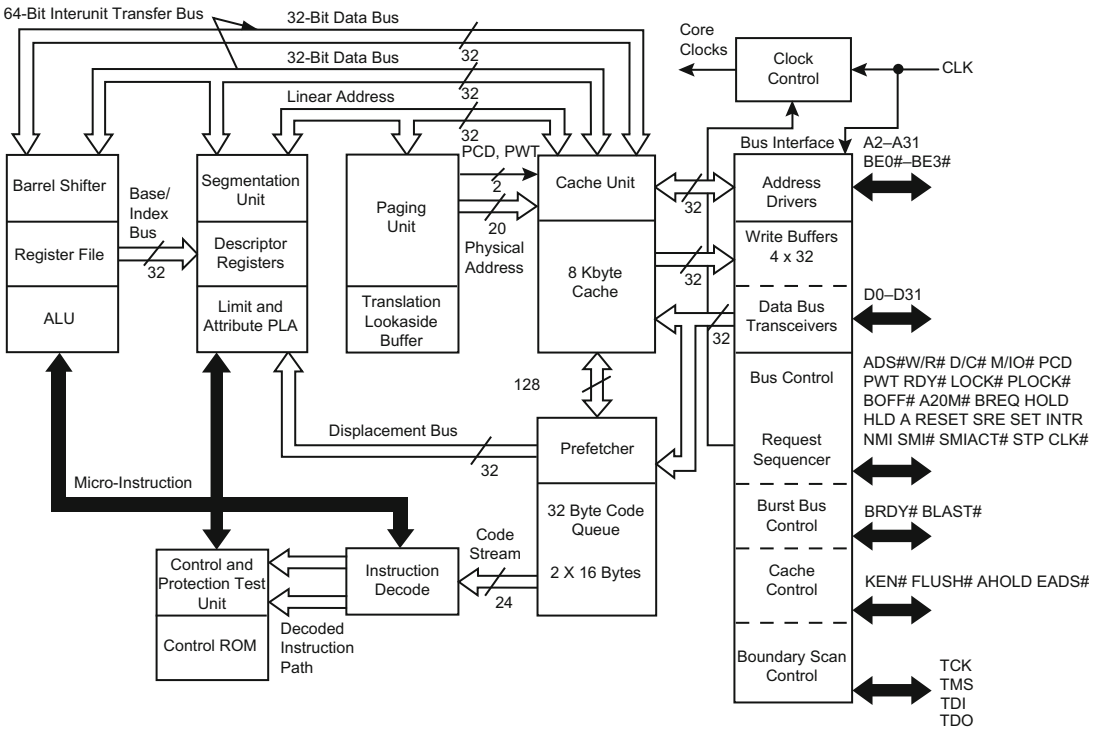


FIGURE 5.1

The function block diagram of the Intel486 GX Processor.

(Courtesy of Intel Corporation.)

(2) Microprocessor

The Pentium series of microprocessors made by Intel, up to now, includes the Pentium Pro, the Pentium II, the Pentium III, and the Pentium 4, etc. As shown in Figure 5.3(a), the Pentium Pro consists of the following basic hardware elements:

(1) Intel Architecture registers

The Intel Architecture register set implemented in the earlier 80x86 microprocessor is extremely small. This small number of registers permits the processor (and the programmer) to keep only a small number of data operands close to the execution units, where they can be accessed quickly. Instead, the programmer is frequently forced to write back the contents of one or more of the processor's registers to memory when additional data operands need to be read from memory in order to be operated on. Later, when the programmer requires access to the original set of data operands, they must again be read from memory. This juggling of data between the register set and memory takes time and exacts a penalty on the performance of the program. Figure 5.3(b) illustrates the Intel Architecture general register set.

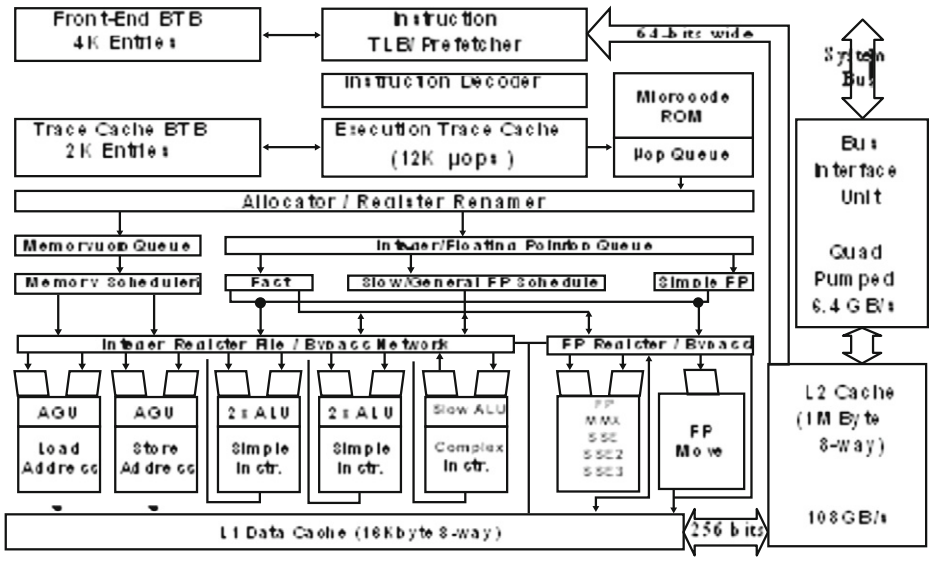


FIGURE 5.2

The function block diagram of the Intel Pentium-4 Processor.

(Courtesy of Intel Corporation.)

(2) External bus unit

This unit performs bus transactions when requested to do so by the L2 cache or the processor core.

(3) Backside bus unit

This unit interfaces the processor core to the unified L2 cache.

(4) Unified L2 cache

Services misses on the L1 data and code caches. When necessary, it issues requests to the external bus unit.

(5) L1 data cache

Services data load and stores requests issued by the load and store execution units. When a miss occurs, it forwards a request to the L2 cache.

(6) L1 code cache

Services instruction fetch requests issued by the instruction prefetcher.

(7) Processor core

The processor logic is responsible for;

1. Instruction fetch.
2. Branch prediction.

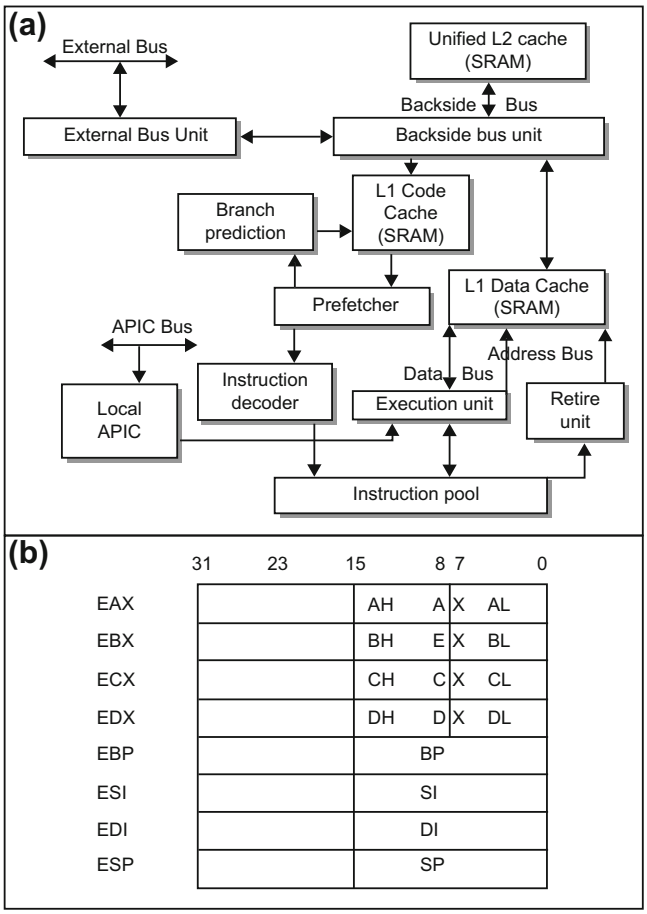


FIGURE 5.3

Intel Pentium II processor: (a) simplified processor block diagram; (b) Intel architecture general register set.
(Courtesy of Intel Corporation.)

3. Parsing of Intel Architecture instruction stream.
4. Decoding of Intel Architecture instructions into RISC instructions that are referred to as micro-ops or u-ops.
5. Mapping accesses for Intel Architecture register set to a large physical register set.
6. Dispatch, execution, and retirement of micro-ops.

(8) Local advanced programmable interrupt controller (APIC) unit

The APIC is responsible for receiving interrupt requests from other processors, the processor local interrupt pins, the APIC timer, APIC error conditions, performance monitor logic, and the IO APIC module. These requests are then prioritized and forwarded to the processor core for execution.

(9) Processor startup

At startup, upon the desertion of reset and the completion of a processor's BIST bit of the configuration register, the processor must wake up and start fetching, decoding, and executing the power-on self test (POST) code from the ROM (read-only memory). The Intel multiprocessing specification (available for download at the Intel developers' website) dictates that the startup code executed by the processor is responsible for detecting the presence of all the components on the processor chipset. When the available components have been detected, the startup code stores this information as a table in nonvolatile memory. According to this specification, both the BIOS code and the POST code are responsible for detecting the presence of and initializing the components. Intel recommends that both the POST code and the BIOS code initialize a predefined RAM location to 1hex. This location is referred to as the central processing unit (CPU) counter.

(10) The fetch, decode, execute engine

At the heart of the processor are the execution units that execute instructions. As shown in Figure 5.3 (a), the processor includes a fetch engine that attempts to properly predict the path of program execution and generates an ongoing series of memory-read operations to fetch the desired instructions.

The high-speed (e.g., 200 MHz or 1.5 GHz) processor execution engine is then bound by the speed of external memory accesses. It should be obvious that it is extremely advantageous to include a very high-speed cache memory on board so that the processor keeps copies of recently used information of both code and data. Memory-read requests generated by the processor core are first submitted to the cache for a lookup before being propagated to the external bus in an event of cache miss.

Pentium processors include both a code cache and a data cache in the level 1 cache. In addition, they include a level 2 cache tightly coupled to the processor core via a private bus. The processor's caches are disabled at power-up time, however. In order to realize the processors' full potential, the caches must be enabled. The steps for the Pentium processors to execute instructions are briefly described below:

1. Fetch Intel Architecture instructions from memory in strict program order.
2. Decode, or translate, them in, also strict program order, into one or more fixed-length RISC instructions known as micro-ops or u-ops.
3. Place the micro-ops into an instruction pool in strict program order.
4. Until this point, the instructions have been kept in original program order. This part of the pipeline is known as the in-order front end. The processor then executes the micro-ops in any order possible as the data and execution units required for each micro-op become available. This is known as the out-of-order portion of pipeline.
5. Finally, the processor commits the results of each micro-op execution to the processor's register set in the order of the original program flow. This is the in-order rear end.

The new Pentium processors implement dynamic execution microarchitecture, a combination of multiple branch prediction, speculation execution, and data flow analysis. These Pentium processors execute MMX (which will be detailed in a subsequent paragraph) technology instructions for enhanced media and communication performance.

Multiple branches predict the flow of the program through several branches. Use of a branch prediction algorithm enables the processor to anticipate jumps in instruction flow. It predicts where the next instruction can be found in memory with a 90% or greater accuracy. This is made possible because

the processor is fetching instructions, and at the same time it is also looking at instructions further ahead in the program.

Data flow analysis analyzes and schedules instructions to be executed in an optimal sequence, independent of the original program order; the processor looks at decoded software instructions and determines whether they are available for processing or whether they are dependent on other instructions.

Speculative execution increases the rate of execution by looking ahead of the program counter and executing instructions that are likely to be needed later. When the processor executes several instructions at a time, it does so using speculative execution. The instructions being processed are based on predicted branches and the results are stored as speculative results. Once their final state can be determined, the instructions are returned to their proper order and committed to permanent machine state.

(11) Processor cache

Figure 5.3(a) provides an overview of the processor's cache, showing its two types; data cache and code cache. The L1 code cache services the requests for instructions generated by the instruction prefetcher (the prefetcher is the only unit that accesses the code cache and it only reads from it, so the code cache is read only), whereas the L1 data cache services memory data read and write requests generated by the processor's execution units when they are executing any instruction that requires a memory data access. The unified L2 cache resides on a dedicated bus referred to as the backside bus. It services misses on the L1 caches, and, in the event of an L2 miss, it issues a transaction request to the external memory. The information is placed in the L2 cache and is also forwarded to the appropriate L1 cache for storage.

The L1 data cache services read and writes requests initiated by the processor execution units. The size and structure of the L1 data cache is processor implementation-specific. As processor core speeds increase, cache sizes will increase to service the greater need for memory access. Each of the data cache's cache banks, is further divided into two banks. When performing a lookup, the data cache views memory as divided into pages equal to the size of one of its cache banks (or ways). Furthermore, it views each memory page as having the same structure as one of its cache ways. The target number is used to index into the data cache directory, and select a set of two entries to compare against. If the target page number matches the tag field in one of the entries in the E, S, or M state given below, it is a cache hit. The data cache has a copy of the target line from the target page. The action taken by the data cache depends on whether or not the data access is a read or a write, the current state of the line, and the rules of conduct defined for this area of memory. Each line storage location within the data cache can currently be in one of four possible states:

1. Invalid state (I); there is no valid line in the entry.
2. Exclusive state (E); the line in the entry is valid, is still the same as memory, and no other processor has a copy of the line in its caches.
3. Shared state (S); the line in the entry is valid, still the same as memory, and one or more processors may also have copies of the line or may not because the processor cannot discriminate between reads by processors and reads performed by other, noncaching entries such as a host/PCI bridge.
4. Modified state (M); the line in the entry is valid, has been updated by this processor since it was read into the cache, and no other processor has a copy of the line in its caches. The line in memory is stable.

The L1 code cache exists only to supply requested code to the instruction prefetcher. The prefetcher issues only read requests to the code cache, so it is a read-only cache. A line stored in the code cache

can only be one of two possible states, valid or invalid, implemented as the S and I states. When a line of code is fetched from memory and is stored in the code cache, it consists of raw code. The designers could have chosen to prescan the code stream as it is fetched from memory and store boundary markers in the code cache to demarcate the boundaries between instructions within the cache line. This would preclude the need to scan the code line as it enters the instruction pipeline, but this would bloat the size of the code cache. Note that the Pentium's code cache stores boundary markers. When performing a lookup, the code cache views memory as divided into pages equal to the size of one of its cache banks (or ways). Furthermore, it views each memory page as having the same structure as one of its cache ways.

(12) MMX technology

Intel's Matrix Math Extensions (MMX) technology is designed to accelerate multimedia and communication applications whilst retaining full compatibility with the original Pentium processor. It contains five architectural design enhancements:

1. New instructions.
2. Single Instruction Multiple Data (SIMD). The new instructions use a SIMD model, operating on several values at a time. Using the 64-bit MMX registers, these instructions can operate on eight bytes, four words, or two double words at once, greatly increasing throughout.
3. More cache. Intel has doubled on-chip cache size to 32k. Hence, more instructions and data can be stored on the chip, reducing the number of times the processor needs to access the slower, off-chip memory area for information.
4. Improved branch prediction. The MMX processor contains four prefetch buffers, which can hold up to four successive code streams.
5. Enhanced pipeline and deeper write buffers. An additional pipeline stage has been added, and four write buffers are shared between the dual pipelines, to improve memory write performance.

MMX technology uses general-purpose, basic, instructions that are fast and easily assigned to the parallel pipelines in Intel processors. By using this general-purpose approach, MMX technology provides performance that will scale well across current and future generations of Intel processors. The MMX instructions cover several functional areas including:

1. Basic arithmetic operations such as add, subtract, multiply arithmetic shift, and multiply add.
2. Comparison operations.
3. Conversation instructions to convert between the new data types pack data together and unpack from small to larger data types.
4. Logical operations such as AND, NOT, OR, and XOR.
5. Shift operations.
6. Data transfer (MOV) instructions for MMX register-to-register transfers, or 64-bit and 32-bit load and store to memory.

The principal data type of the MMX instruction set is the packed, fixed-point integer, where multiple integer words are grouped into single 64-bit quantities. These quantities are moved to the 64-bit MMX registers. The decimal point of the fixed-point values is implicit, and is left for the programmer to control for maximum flexibility. Arithmetic and logical instructions are designed to support the different packed integer data types. These instructions have a different op code for each data type

supported. As a result, the new MMX technology instructions are implemented with 57 op codes. The supported data types are signed and unsigned fixed-point integers, bytes, words, double words, and quad words. The four MMX technology data types are:

1. packed bytes: 8 bytes packed into one 64-bit quantity;
2. packed word: four 16-bit words packed into one 64-bit quantity;
3. packed double word: two 32-bit double words packed into one 64-bit quantity;
4. quad word: one 64-bit quantity.

From the programmer's point of view, there are eight new MMX registers (MM0 MM7), along with new instructions that operate on these registers. But to avoid adding new states, these registers are mapped onto the existing floating-point registers (FP0 FP7). When a multitasking operating system (or application) executes an FSAVE instruction, e.g. to save state, the contents of MM0 MM7 are saved in place of FP0 FP7 if MMX instructions are in use.

Detecting the existence of MMX technology on an Intel microprocessor is done by executing the CPUID instruction and checking a set bit. Therefore, when installing or running, a program can query the microprocessor to determine whether or not MMX technology is supported, and proceed according to the result.

(3) Internal bus system

Figures 5.1 and 5.3(a) show that the internal bus systems of an Intel microprocessor, based on their functions, comprises the types below, each of them monitored by a corresponding bus controller or bus unit:

1. Backside bus.
2. Displacement bus.
3. APIC bus.
4. Cache buses, divided into data bus and address bus.
5. CPUs' cluster bus.

(4) Memories

Memories on microprocessor chipsets can be classified into two groups; prime (system or main) memory and storage memory. The read/write memory (R/WM) and read-only memory (ROM) are examples of prime memory the memory the microprocessor uses in executing and storing programs. This memory should be able to keep up with the execution speed of the microprocessor, so it should be random access memory, meaning that the microprocessor should be able to access information from any register with the same speed (independent of its place in the chip). The size of a memory chip is specified in terms of bits. For example, a 1k memory chip means it can store 1k (1024) bits (not bytes). On the other hand, system memory in a PC is specified in bytes so 4M memory means it is 4 megabytes in size.

The other group is the storage memory, which uses devices such as magnetic disks and tapes (see Figure 5.4). This memory is used to store programs and results after the completion of program execution. Such storage is nonvolatile, meaning information remains intact even if the system power is turned off. The microprocessor cannot access programs stored in these devices directly; they need to be copied into the R/W prime memory first. Hence, the size of the prime memory, determines how large

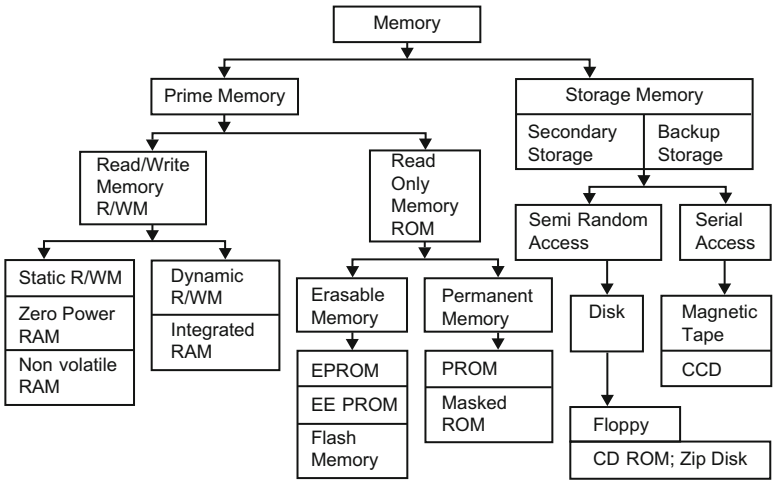


FIGURE 5.4

The classification of microprocessor memory.

a program the system can process. Storage memory is unlimited in size; when one disk is full, the next one can be used.

Figure 5.4 shows that these are also two types of storage memory; secondary storage and backup storage, which includes devices such as disks, magnetic tapes, etc. Prime (system) memory is divided into two main groups: read/write memory (R/WM) and read-only memory (ROM); each group includes several different types of memory, as discussed below.

(1) Read/write memory (R/WM)

As the name suggests, the microprocessor can write into or read from this memory; it is popularly known as random access memory (RAM). It is used primarily for information that is likely to be altered, such as writing programs or receiving data. This memory is volatile, meaning that when the power is turned off, all the contents are lost. Two types of R/W memories, static and dynamic are available; they are described in the following paragraphs.

1. Static memory (SRAM). This memory is made up of flip-flops, and it stores each bit as a voltage. Each memory cell requires six transistors, so the memory chip has low density but high speed. SRAM, known as cache memory, resides on the processor chip. A high-speed cache is also included, external to the processor to improve the performance of a system.
2. Dynamic memory (DRAM). This memory is made up of MOS transistor gates, and it stores each bit as a charge. In dynamic memory, stored information needs to be read and then written again every few milliseconds. It is generally used when system memory is at least 8k; for smaller systems, static memory is more appropriate. Various techniques are used to increase the speed of DRAM, which have resulted in the production of high-speed memory chips, such as Extended Data Out (EDO), Synchronous DRAM (SDRAM), and Rambus DRAM (RDRAM).

(2) Read-only memory (ROM)

ROM is nonvolatile memory; it retains stored information even if the power is turned off. This memory is used for programs and data that need not be altered. As the name suggests, the information can only be read, which means once a bit pattern is stored, it is permanent or at least semi-permanent. The permanent group also includes two types of memory; masked ROM and PROM, as does the semi-permanent group; EPROM and EE-PROM, as shown in Figure 5.4. The five types of ROM masked ROM, PROM, EPROM, EE-PROM, and flash memory are described in the following paragraphs.

1. Masked ROM. In this ROM, a bit pattern is permanently recorded by the masking and metallization process in chip manufacture. It is an expensive and specialized process, but economical for large production quantities.
2. Programmable read-only memory (PROM). This memory has nichrome or polysilicon wires arranged in a matrix, which can be viewed functionally as diodes or fuses. This memory can be programmed by the user via a special PROM programmer that selectively burns the fuses according to the bit pattern to be stored. The process is known as "burning the PROM", and the information stored is permanent.
3. Erasable, programmable, read-only memory (EPROM). This memory stores a bit by charging the floating gate of an FET. Information is stored by using an EPROM programmer, which applies high voltages to charge the gate. All the information can be erased by exposing the chip to ultraviolet light through its quartz window, and the chip can be reprogrammed which makes it ideally suited for product development and experimental projects. The disadvantages of EPROM are (1) it must be taken out of the circuit to erase it, (2) the entire chip must be erased, and (3) the erasing process can take 15 or 20 minutes.
4. Electrically erasable PROM (EE-PROM). This memory is functionally similar to EPROM, except that information can be altered by using electrical signals at register level, rather than by erasing it all. This has an advantage in field and remote-control applications. In microprocessor systems, software updating is a common occurrence. If EE-PROMs are used, they can be updated remotely from a central computer. This memory also includes Chip Erase mode, whereby the entire chip can be erased in 10 milliseconds rather than the 20 minutes needed to erase an EPROM.
5. Flash memory. This is a variation of EE-PROM is now becoming popular. The major difference between flash memory and EE-PROM is in the erasure procedure; the EE-PROM can be erased at a register level, but the flash memory chip must be erased either in its entirety or at sector (block) level. These memory chips can be erased and programmed at least a million times.

In a microprocessor-based device, programs are generally written in ROM, and data that are likely to vary are stored in R/W memory. In addition to static and dynamic R/W memory, other options are also available in memory devices. Examples include zero power RAM, nonvolatile RAM, and integrated RAM.

Zero power RAM is a CMOS read/write memory with battery backup built internally. It includes lithium cells and voltage-sensing circuitry. When the external power supply voltage falls below 3 V, the power-switching circuitry connects to the lithium battery; thus, this memory provides the advantages of R/W and read-only memory.

Nonvolatile RAM is a high-speed, static, R/W memory array backed up, bit for bit, by EE-PROM for nonvolatile storage. When power is about to go off, the contents of R/W memory are stored in the EE-PROM by activating the store signal or the memory chip, then stored data can be read into the R/W memory segment when the power is resumed. This memory chip combines the flexibility of static R/W memory with the nonvolatility of EE-PROM.

Integrated RAM (iRAM) is a dynamic memory with the refreshed circuitry built on a chip. For the user, it is similar to static R/W memory, giving the advantages of dynamic memory without having to build the external refresh circuitry.

(5) Input/output pins

To allow for easy upgrades and to save space, the 80486 and Pentium processors are available in a pin-grid array (PGA) form. For all the Intel microprocessors, their PGA pin-out lists are provided in the corresponding Intel specifications. A 168-pin 80486 GX block is illustrated in Figure 5.1; it can be seen that the 80486 processor has a 32-bit address bus (A0-A31) and a 32-bit data bus (D0-D31). Table 5.1 defines how the 80486 control signals are interpreted. Table 5.2 lists the main input (I) and output (O) pins of Intel 80486 processor.

(6) Interrupt system

In the Intel microprocessors, the interrupt line types can be interrupt request pin (INTR), nonmaskable interrupt request pin (NMI), and system reset pin (RESET), all of which are high signals. The INTR pin is activated when an external device, such as a hard disk or a serial port, wishes to communicate with the processor. This interrupt is maskable can be ignored if necessary. The NMI pin is a non-maskable interrupt and so is always acted on. When it becomes active the processor calls the non-maskable interrupt service routine. The RESET pin signal causes a hardware reset and is normally activated when the processor is powered up.

5.1.2 Microprocessor interrupt operations

The interrupt I/O is a process of data transfer, whereby an external device or peripheral can inform the processor that it is ready for communication and it requests attention. The process is initiated by an external device and is asynchronous, meaning that it can be initiated at any time without reference to

Table 5.1 Intel 80486 Processor Control Signal

M/IO	D/C	W/R	Description
0	0	0	Interrupt acknowledge sequence
0	0	1	STOP/special bus cycle
0	1	0	Reading from an I/O port
0	1	1	Writing to an I/O port
1	0	0	Reading a instruction from memory
1	0	1	Reserved
1	1	0	Reading data from memory
1	1	1	Writing data to memory

Table 5.2 Intel 80486 Main Pin Connections

Pin	Specification
(1) A2-A31 (I/O)	The 30 most significant bits of the address bus
(2) A20M (I)	When active low, the processor internally masks the address bit A20 before every
(3) ADS (O)	Indicates that the processor has valid control signals and valid address signals
(4) AHOLD (I)	When active, a different bus controller can have access to the address bus. This is typically used in a multiprocessor system
(5) BE0 – BE3 (O)	The byte enable lines indicate which of the bytes of the 32-bit data bus is active
(6) BLAST (O)	It indicates that the current burst cycle will end after the next BRDY signal
(7) BOFF (I)	The backoff signal informs the processor to deactivate the bus on the next clock cycle
(8) BRDY (I)	The burst ready signal is used by an addressed system that has sent data on the data bus or read data from the bus
(9) BREQ (O)	It indicates that the processor has internally requested the bus
(10) BS16, BS8 (I)	The BS16 signal indicates that a 16-bit data bus is used; the BS8 signal indicates that an 8-bit data bus is used. If both are high, then a 32-bit data bus is used
(11) DP0 – DP3 (I/O)	The data parity bits give a parity check for each byte of the 32-bit data bus. The parity bits are always even parity
(12) EADS (I)	Indicates that an external bus controller has put a valid address on the address bus
(13) FERR (O)	Indicates that the processor has detected an error in the internal floating-point unit
(14) FLUSH (I)	When it is active the processor writes the complete contents of the cache to memory
(15) HOLD, HOLDA (I/O)	The bus hold (HOLD) and acknowledge (HOLDA) are used for bus arbitration and allow other bus controllers to take control of the buses
(16) IGNNE (I)	When active the processor ignores any numeric errors
(17) INTR (I)	External devices to interrupt the processor use the interrupt request line
(18) KEN (I)	This signal stops caching of a specific address
(19) LOCK (O)	If it is active, the processor will not pass control to an external bus controller when it receives a HOLD signal
(20) M/IO, D/C, W/R (O)	See Table 5.1
(21) NMI (I)	The nonmaskable interrupt signal causes an interrupt 2
(22) PCHK (O)	If it is set active then a data parity error has occurred
(23) PLOCK (O)	The active pseudo lock signal identifies that the current data transfer requires more than one bus cycle
(24) PWT, PCD (O)	The page write-through (PWT) and page cache disable (PCD) are used with cache control
(25) RDY (I)	When it is active, the addressed system has sent data on the data bus or read data from the bus
(26) RESET (I)	If the reset signal is high for more than 15 clock cycles, then the processor will reset itself

the system clock. However, the response to an interrupt request is directed or controlled by the microprocessor. Unlike the polling technique, interrupt processing allows a program, or an external device, to interrupt the current task. An interrupt can be generated by hardware (hardware interrupt) or by software (software interrupt). At this point an interrupt service routine (ISR) is called. For a hardware interrupt, the ISR then communicates with the device and processes data, after which it returns to the original program. A software interrupt causes the program to interrupt its execution and go to an ISR. Software interrupts include the processor-generated interrupts that normally occur either when a program causes a certain type of error, or if it is being used in debug mode. In the latter case the program can be made to break from its execution when a breakpoint occurs. Software interrupts, in most cases, do not require the program to return when the ISR task is complete. Apart from this difference, both software and hardware interrupts use the same mechanisms, methodologies, and processes to handle interrupts.

Interrupt requests are classified as maskable interrupt and nonmaskable interrupt. The microprocessor can ignore or delay a maskable interrupt request if it is performing some critical task; however, it must respond to a nonmaskable interrupt immediately.

(1) Interrupt process

The operation of an interrupt depends upon the system mode in which when the interrupt occurs, either real mode or protection mode.

(1) The operation of a real mode interrupt

When the microprocessor completes executing the current instruction, it determines whether an interrupt is active by checking (1) instruction executions, (2) single step, (3) NMI pin, (4) coprocessor segment overrun, (5) INTR pin, and (6) INT instruction, in the order presented. If one or more of these interrupt conditions are present:

1. The contents of the flag register are pushed onto the stack.
2. Both the interrupt (IF) and trap (TF) flags are cleared. This disables the INTR pin and the trap or single-step feature.
3. The contents of the code segment register (CS) are pushed onto the stack.
4. The contents of the instruction pointer (IP) are pushed onto the stack.
5. The interrupt vector contents are fetched, and then placed into both IP and CS so that the next instruction executes the ISR addressed by the vector.

Whenever an interrupt is accepted, the microprocessor stacks the contents of the flag register, CS and IP; clears both IF and TF and then jumps to the procedure addressed by the interrupt vector. After the flags are pushed onto the stack, IF and TF are cleared. These flags are returned to the state prior to the interrupt when the IRET instruction is encountered at the end of the ISR. Therefore, if interrupts were enabled prior to the ISR, they are automatically re-enabled by the IRET instruction at the end of the interrupt service routine.

The return address (stored in CS and IP) is pushed onto the stack during the interrupt. Sometimes, the return address points to the next instruction in the program, and sometimes it points to the instruction or point in the program where the interrupt occurred. Interrupt type numbers 0, 5, 6, 7, 8, 10, 11, 12, and 13 push a return address that points to the offending instruction, instead of to the next instruction in the program. This allows the ISR to retry the crashed instruction in certain error cases.

Some of the protected mode interrupts (type 8, 10, 11, 12, and 13) place an error code on the stack following the return address. This code identifies the selector that caused the interrupt. If no selector is involved, the error code is 0.

(2) The operation of a protected mode interrupt

In protected mode, interrupts have exactly the same assignments as in real mode, but the interrupt vector table is different. In place of interrupt vectors, protected mode uses a set of 256 interrupt descriptors that are stored in an interrupt descriptor table (IDT), normally 256 × 8 (2k) bytes long, with each descriptor containing 8 bytes. It is located at any memory location in the system by the IDT address register (IDTR). Each entry in the IDT contains the address of the ISR, in the form of a segment selector and a 32-bit offset address. It also contains the P bit (present) and DPL bits, which describe the privilege level of the interrupt.

Real mode interrupt vectors can be converted into protected mode interrupts by copying the interrupt procedure addresses from the interrupt vector table and converting them to 32-bit offset addresses that are stored in the interrupt descriptors. A single selector and segment descriptor can be placed in the global descriptor table that identifies the first 1M byte of memory as the interrupt segment.

Other than the IDT and interrupt descriptors, the protected mode interrupt functions like the real mode interrupt. They return from both interrupts by using the IRET or IRETD instruction. The only difference is that in protected mode the microprocessor accesses the IDT instead of the interrupt vector table.

(3) Interrupt flag bits

The interrupt flag (IF) and trap flag (TF) are both cleared after the contents of the flag register are stacked during an interrupt. When the IF bit is set, it allows the INTR pin to cause an interrupt; when the IF bit is cleared, it prevents the INTR pin from causing an interrupt. When IF = 1, it causes a trap interrupt (interrupt type number 1) to occur after each instruction executes. This is why we often call trapping a single step. When TF = 0, normal program execution occurs. The interrupt flag is set and cleared by the STI and CLI instructions, respectively. There are no special instructions that set or clear the trap flag.

(2) Interrupt vectors

The interrupt vectors and vector table are crucial to the understanding of hardware and software interrupts. Interrupt vectors are addresses that inform the interrupt handler as to where to find the ISR (interrupt service routine, also called interrupt service procedure). All interrupts are assigned a number from 0 to 255, with each of these interrupts being associated with a specific interrupt vector.

The interrupt vector table is normally located in the first 1024 bytes of memory at addresses 000000H 0003FFH. It contains 256 different interrupt vectors. Each vector is 4 bytes long and contains the starting address of the ISR. This starting address consists of the segment and offset of the ISR. Figure 5.5 illustrates the interrupt vector table used for the Intel microprocessors. Remember that in order to install an interrupt vector (sometimes called a hook), the assembler must address absolute memory.

In an interrupt vector table, the first five interrupt vectors are identical in all Intel microprocessor family members, from the 8086 to the Pentium. Other interrupt vectors exist for the 80286 that are upward-compatible to 80386, 80486, and Pentium to Pentium 4, but not downward-compatible to the

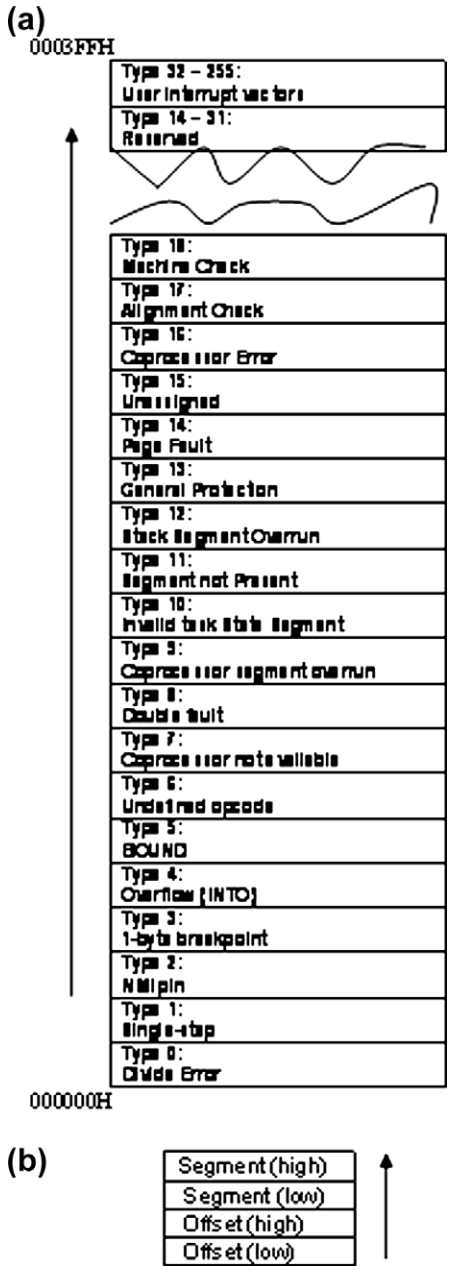


FIGURE 5.5

(a) The interrupt vector table for the Intel microprocessor; (b) the contents of an interrupt vector.

8086 or 8088. Intel reserves the first 32 interrupt vectors for its use in various microprocessor family members. The last 224 vectors are available as user interrupt vectors.

(3) Interrupts service routine (ISR)

The interrupts of the entire Intel family of microprocessors include two hardware pins that request interrupts (INTR pin and NMI pin), and one hardware pin (INTA) that acknowledges the interrupt requested through INTR. In addition to these pins, the Intel microprocessor also has software interrupt instructions: INT, INTO, INT 3, and BOUND. Two flag bits, IF (interrupt flag) and TF (trap flag), are also used with the interrupt structure and with a special return instruction IRET (or IRETD in the 80386, 80486, or Pentium-Pentium 4).

(1) Software interrupts

Intel microprocessors provide five software interrupt instructions: BOUND, INTO, INT, INT 3, and IRET. Of these five instructions, INT and INT 3 are very similar, BOUND and INTO are conditional, and IRET is a special interrupt return instruction.

The INT *n* instruction calls the ISR that begins at the address represented by the vector number *n*. The only exception to this is the "INT 3" instruction, a 1-byte instruction, which is used as breakpoint-interrupt, because it is easy to insert a 1-byte instruction into a program. As mentioned previously, breakpoints are often used to debug faulty software.

The BOUND instruction, which has two operands, compares a register with two words of memory data. The INTO instruction checks the overflow flag (OF); If OF = 1, the INTO instruction calls the ISR whose address is stored in interrupt vector type number 4. If OF = 0, then the INTO instruction performs no operation and the next sequential instruction in the program executes.

The IRET instruction is a special return instruction used to return for both software and hardware interrupts. The IRET instruction is much like a "far RET" because it retrieves the return address from the stack. It is unlike the "near return" because it also retrieves a copy of the flag register. An IRET instruction removes six bytes from the stack: two for the IP, two for CS, and two for flags. In the 80386 to the Pentium 4, there is also an IRETD instruction, because these microprocessors can push the EFLAG register (32 bit) on the stack, as well as the 32-bit EIP in protected mode. If operated in the real mode, we use the IRET instruction with the 80386 to Pentium 4 microprocessors.

(2) Hardware interrupts

The microprocessor has two hardware inputs; nonmaskable interrupt (NMI) and interrupt request (INTR). Whenever the NMI input is activated, a type 2 interrupt occurs because NMI is internally decoded. The INTR input must be externally decoded to select a vector. Any interrupt vector can be chosen for the INTR pin, but we usually use an interrupt type number between 20H and FFH. Intel has reserved interrupts 00H through 1FH for internal and future expansion. The INTA signal is also an interrupt pin on the microprocessor, but it is an output that is used in response to the INTR input to apply a vector-type number to the data bus connections D7 D0.

The NMI is an edge-triggered input that requests an interrupt on the positive edge (0-to-1 transition). After a positive edge, the NMI pin must remain logic 1 until it is recognized by the microprocessor. The NMI input is often used for parity errors and other major system faults, such as power failure. Power failures are easily detected by monitoring the AC (alternating current) power line and causing an NMI interrupt whenever AC power drops out.

The interrupt request input (INTR) is level-sensitive, which means that it must be held at logic 1 level until it is recognized. The INTR pin is set by an external event and cleared inside the ISR. This input is automatically disabled once it is accepted by the microprocessor and reenabled by the IRET instruction at the end of the ISR. The microprocessor responds to the INTR input by pulsing the INTA output in anticipation of receiving an interrupt vector-type number on data bus connection D7 D0. There are two INTA pulses generated by the system that are used to insert the vector-type number on the data bus.

5.1.3 Microprocessor unit input/output rationale

The I/O devices can be interfaced with a microprocessor using two techniques: isolated I/O (also called peripheral-mapped I/O) and memory-mapped I/O. The process of data transfer in both is identical. Each device is assigned a binary address, called a device address or port number, through its interface circuit. When the microprocessor executes a data transfer instruction for an I/O device, it places the appropriate address on the address bus, sends the control signals, enables the interfacing device, and then transfers data. The interface device is like a gate for data bits, which is opened by the microprocessor whenever it intends to transfer data.

(1) Basic input/output techniques

As previously mentioned, there are two main methods of communicating with external equipment either the equipment is mapped into the physical memory and given a real address on the address bus of the microprocessor (memory-mapped I/O), or it is mapped into a special area of input/output memory (isolated I/O). Devices mapped into memory are accessed by reading or writing to its physical address. Isolated I/O provides ports that are gateways between the interface device and the processor. They are isolated from the system using a buffering system, and are accessed by four machine code instructions: IN, INS, OUT, OUTS. The IN (INS) instruction inputs a byte, or a word, and the OUT (OUTS) instruction outputs a byte, or a word. A high-level compiler interprets the equivalent high-level functions and produces machine code that uses these instructions.

Figure 5.6 shows the two methods. This figure also tells us that devices are not directly connected to the address and data bus because they may use part of the memory that a program uses or they could cause a hardware fault. This device interprets the microprocessor signals and generates the required memory signals. Two main output lines differentiate between a read and a write operation (R/W) and between direct and isolated memory access (M/IO). The R/W line is low when data are being written to memory and high when data are being read. When M/IO is high, direct memory access is selected, and when low, the isolated memory is selected.

(1) Isolated I/O

The most common I/O transfer technique used in the Intel microprocessor-based system is isolated I/O. The term "isolated" describes how the I/O locations are isolated from the memory system in a separate I/O address space. The addresses for isolated I/O devices, called ports, are separate from the memory, hence the user can expand the memory to its full size without using any of the memory space reserved for I/O devices. A disadvantage of isolated I/O is that the data transferred between I/O and the microprocessor must be accessed by the IN, INS, OUT, and OUTS instructions. Separate control signals for the I/O space are developed (using M/IO and R/W), which indicate an I/O read (IORC) or an I/O write (IOWC) operation. These signals indicate that an I/O port address, which appears on the

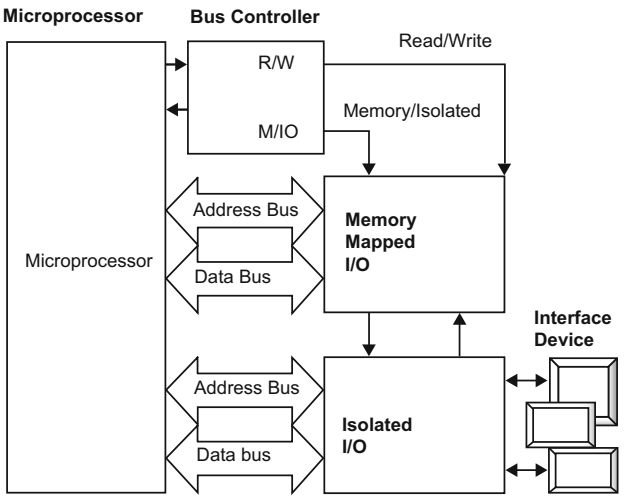


FIGURE 5.6

Access memory-mapped and isolated I/O.

address bus, is used to select the I/O device. In the personal computer, isolated I/O ports are used for controlling peripheral devices such as direct memory access (DMA) controller, NMI reset, game I/O adaptor, floppy disk controller, second serial port (COM2), and primary serial port (COM1). An 8-bit port address is used to access devices located on the system board, such as the timer and keyboard interface, while a 16-bit port is used to access serial and parallel ports as well as video and disk drive system.

(2) Memory-mapped I/O

Interface devices can map directly onto the system address and data bus. Unlike isolated I/O, memory-mapped I/O does not use the IN, INS, OUT, or OUTS instructions. Instead, it uses any instruction that transfers data between the microprocessor and memory. A memory-mapped I/O device is treated as a memory location in the memory map, the main advantage of this being that any memory transfer instruction can be used to access the I/O device. The main disadvantage is that a portion of the memory system is used as the I/O map, which reduces the amount of usable memory available for applications.

In a PC-compatible system the address bus is 20 bits wide, from address 00000h to FFFFFh (1MB). Figure 5.7 gives a typical memory allocation in PC.

(2) Basic input/output interfaces

The basic input device is a set of three-state buffers, and the basic output device is a set of data latches. The term IN refers to moving data from the I/O device into the microprocessor, and the term OUT refers to moving data out of the microprocessor to the I/O device.

Many I/O devices accept or release information at a much slower rate than the microprocessor. Another method of I/O control, called handshaking or polling, synchronizes the I/O device with the

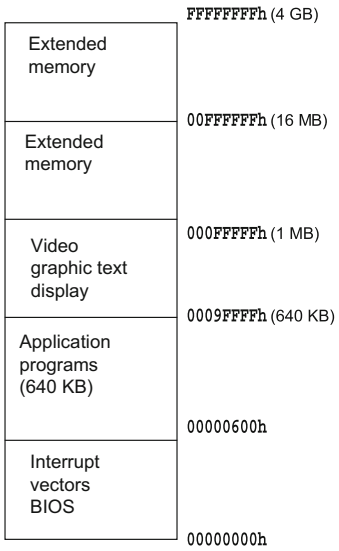


FIGURE 5.7

Typical PC memory map.

microprocessor. An example of a device that requires handshaking is a parallel printer that prints 100 characters per second (CPS). It is obvious that the microprocessor can send more than 100 CPS to the printer, so a handshaking must be used to slow the microprocessor down to match the speed of printer.

(1) The basic input interface

Three-state buffers 74ALS244 are used to construct the 8-bit input port depicted in Figure 5.8(a). The external TTL data (simple toggle switches in this example) are connected to the inputs of the buffers. The outputs of the buffers connect to the data bus. The exact data bus connections depend on the version of the microprocessor. For example, the 8088 has data bus connections D7 D0, the 80486 has D31 D0, and the Pentium to Pentium 4 have D63 D0. The circuit of Figure 5.8(a) allows the microprocessor to read the contents of the eight switches that connect to any 8-bit section of the data bus when the select signal SEL becomes logic 0. Thus, whenever the IN instruction executes, the contents of the switches are copied into the AL register.

When the microprocessor executes an IN instruction, the I/O port address is decoded to generate the logic 0 on SEL. A 0 placed on the output control inputs (1G and 2G) of the 74ALS244 buffer causes the data input connections (A) to be connected to the data input (Y) connections. If a logic 1 is placed on the output control inputs of the 74ALS244 buffer, the device enters the three-state high-impedance mode that effectively disconnects the switches from the data bus.

The basic input circuit is not optional and must appear any time that input data are interfaced to the microprocessor. Sometimes it appears as a discrete part of the circuit, as shown in Figure 5.8(a); sometimes it is built into a programmable I/O device.

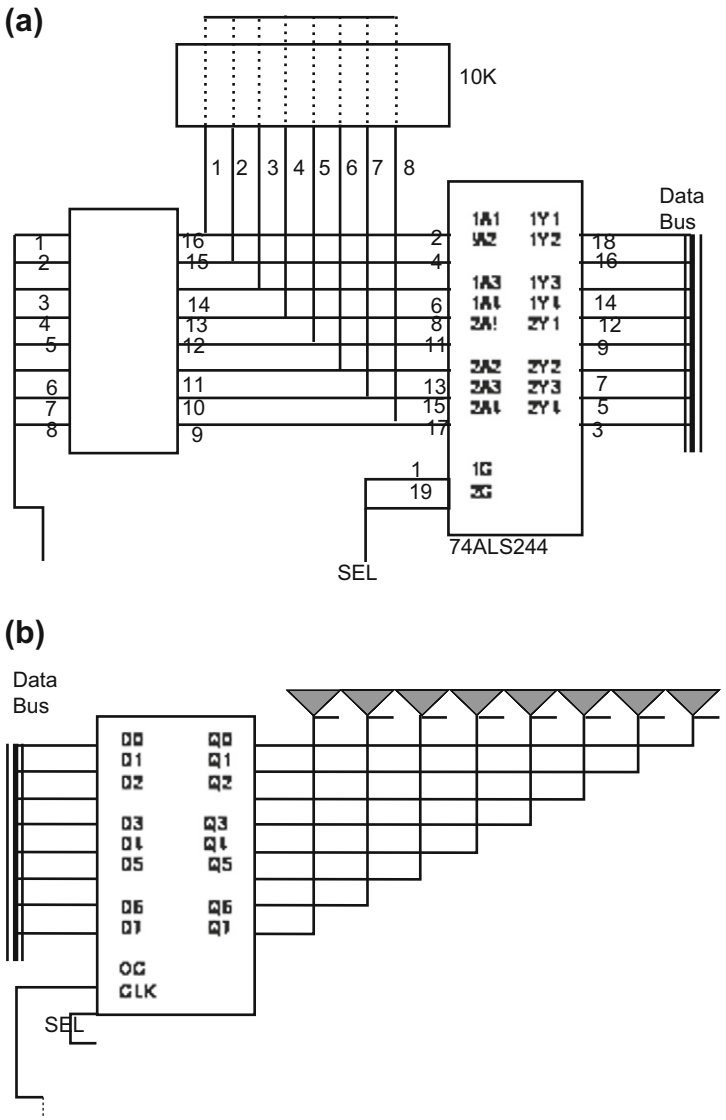


FIGURE 5.8

The basic input and output interfaces. (a) The basic input interface illustrating the connection of eight switches. Note that the 74ALS244 is a three-state that controls the application of switch data to the data bus. (b) The basic output interface connected to a set of LED displays.

It is possible to interface 16- or 32-bit data to various versions of the microprocessor, but this is not nearly as common as using 8-bit data. To interface 16 bits of data, the circuit in Figure 5.8(a) is doubled to include two 74ALS244 buffers that connect 16 bits of input data to the 16-bit data bus. To interface 32 bits of data, the circuit is expanded by a factor of 4.

(2) The basic output interface

The basic output interface receives data from the microprocessor and must usually hold them for some external device. Its latches or flip-flops, like the buffers found in the input device, are often built into the I/O device.

Figure 5.8(b) shows how eight simple light-emitting diodes (LEDs) connect to the microprocessor through a set of eight data latches. The latch stores the number that is output by the microprocessor vice the data bus so that the LED can be lit with any 8-bit binary number. Latches are needed to hold the data, because when the microprocessor executes an OUT instruction, the data are present on the data bus for less than 1.0 μ s. Without a latch, the viewer would never see the LED illuminate.

When the OUT instruction executes, the data from AL, AX, or EAX are transferred to the latch via the data bus. Here, the D inputs of a 74ALS374 octal latch are connected to the data bus to capture the output data, and the Q outputs of the latch are attached to the LED. When a Q becomes a logic 0, the LED lights. Each time that the OUT instruction executes, the SEL signal to the latch activates, capturing the data output to the latch from any 8-bit section of the data bus. The data are held until the next OUT instruction executes. Thus, whenever the output instruction is executed in this circuit, the data from the AL register appear on the LED.

5.1.4 Microprocessor unit bus system operations

This subsection uses the peripheral component interconnect (PCI) bus to introduce the microprocessor unit bus system operations. The PCI bus has been developed by Intel for its Pentium processors. This technique can be populated with adaptors requiring fast accesses to each other and/or system memory, and that can be accessed by the processor at speeds approaching that of the processor's full native bus. A PCI physical device package may take the form of a component integrated onto the system board, or may be implemented on a PCI add-in card. Each PCI package (referred to in the specification as a device) may incorporate from one to eight separate functions. A function is a logical device, which contains its own, individually addressable configuration space, 64 double words in size. Its configuration registers are implemented in this space. Using these registers, the configuration software can automatically detect the presence of a function, determine its resource requirements including memory space, I/O space, interrupt lines, etc., and can then assign resources to the function that are guaranteed not to conflict with the resources assigned to other devices.

(1) Bus operations

The PCI bus operates in multiplexing mode (also called normal mode) and/or in burst mode. In the former, the address and data lines are used alternatly. First, the address is sent, followed by a data read or write. Unfortunately, this mode requires two or three clock cycles for a single transfer of an address followed by a read or write cycle. The multiplex mode obviously slows down the maximum transfer rate.

Additionally, a PCI bus can be operated in burst mode. A burst transfer consists of a single address phase followed by two or more data phases. In this mode, the bus master only has to arbitrate

for bus ownership once. The start addresses and transaction type are issued during the address phase. All devices on the bus latch the address and transaction type and decode them to determine which the target device is. The target device latches the start address into an address counter and is responsible for incrementing the address from data phase to data phase. Figure 5.9 shows an example of burst data transfer.

There are two participants in every PCI burst transfer; the initiator and the target. The initiator, or bus master, is the device that initiates a transfer. The target is the device currently addressed by the initiator for the purpose of performing a data transfer. PCI initiator and target devices are commonly referred to as PCI-compliant agents in the specifications. It should be noted that a PCI target may be designed such that it can only handle single data phase transactions. When a bus master attempts to perform a burst transaction, the target forces the master to terminate the transaction at the completion of the first data phase. The master must rearbitrate for the bus to attempt resumption of the burst when the next data phase completes. In each burst transfer; (1) the address and the transfer type are output during the address phase; (2) a data object may then be transferred during each subsequent data phase.

Assuming that neither the initiator nor the target device inserts wait states in each data phase, a data object may be transferred on the rising edge of each PCI clock cycle. At a PCI bus clock frequency of 33 MHz, a transfer rate of 132 MB/s may be achieved. A transfer rate of 264 MB/s may be achieved in a 64-bit implementation when performing 64-bit transfers during each data phase.

(1) Address phase

Refer to Figure 5.10. Every PCI transaction (with the exception of a transaction using 64-bit addressing) starts off with an address phase one PCI clock period in duration. During the address phase, the initiator identifies the target device and the type of transaction (also referred to as command type). The target device is identified by driving a start address within its assigned range onto the PCI

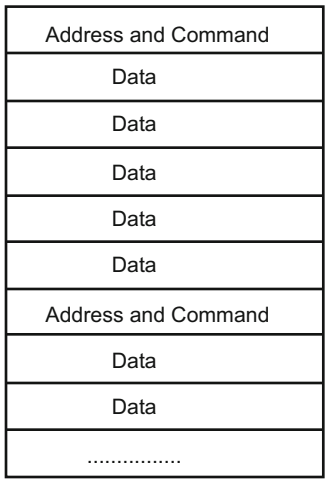


FIGURE 5.9

Example of the burst data transfer.

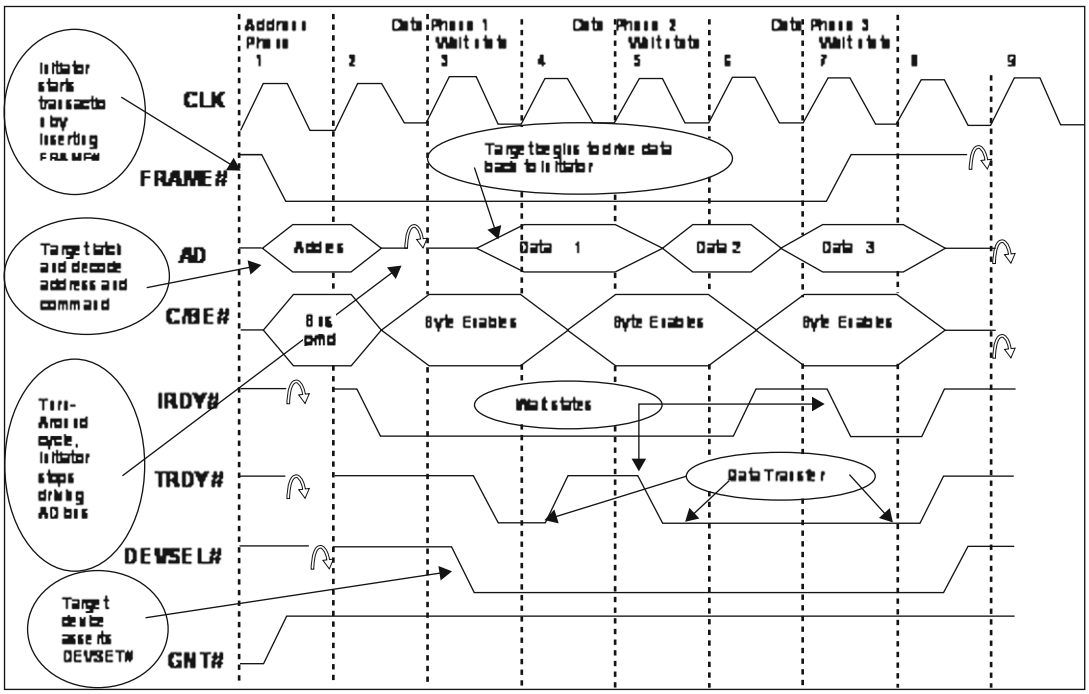


FIGURE 5.10

Typical PCI bus transactions.

address and data bus. At the same time, the initiator identifies the type of transaction by driving the command type onto the 4-bit-wide PCI Command/Byte Enable bus. The initiator also asserts the FRAME# signal to indicate the presence of a valid start address or transaction type on the bus. Since the initiator only presents the start address and command for one PCI clock cycle, it is the responsibility of every PCI target device to latch the address and command on the next rising edge of the clock so that it may subsequently be decoded.

By decoding the address latched from the address bus, and the command type latched from the Command/Byte Enable bus, a target device can determine whether it is being addressed or not, and the type of transaction that is in progress. It is important to note that the initiator only supplies a start address to the target during the address phase. Upon completion of the address phase, the address or data bus becomes the data bus for the duration of the transaction and is used to transfer data in each of the data phases. It is the responsibility of the target to latch the start address, and to autoincrement it to point to the next group of locations during each subsequent data transfers.

(2) Data phase

Refer to Figure 5.10. The data phase of a transaction is the period during which a data object is transferred between the initiator and the target. The number of data bytes to be transferred during a data phase is determined by the number of Command/Byte Enable signals that are asserted by the

initiator during the data phase. Each data phase is at least one PCI clock period in duration. Both the initiator and the target must indicate that they are ready to complete a data phase, or else it is extended by a wait state that is one PCI CLK period in duration. The PCI bus defines ready signal lines to be used by both the initiator (IRDY#) and the target (TRDY#) for this purpose. The initiator does not issue a transfer count to the target. Rather, in each data phase it indicates whether it is ready to transfer the current data item and, if it is, whether it is the final data item. FRAME# is inserted at the start of the address phase and remains inserted until the initiator is ready (inserts IRDY#) to complete the final data phase. When the target samples IRDY# are inserted and FRAME# are not inserted, it realizes that this is the final data phase.

Refer to Figure 5.10. The initiator indicates that the last data transfer (of a burst transfer) is in progress by uninserting FRAME# and inserting IRDY#. When the last data transfer has been completed, the initiator returns the PCI bus to the idle state by uninserting its ready line (IRDY#). If another bus master had previously been granted ownership of the bus by the PCI bus arbiter and was waiting for the current initiator to surrender the bus, it can detect that the bus has returned to the idle state by detecting FRAME# and IRDY# are both uninserted on the same rising edge of the PCI clock.

(2) Bus system arbitration

Bus masters are devices on a PCI bus that are allowed to take control of that bus. This is done by a component named a bus arbiter, which usually integrated into the PCI chipset. Specifically, it is typically integrated into the host/PCI or the PCI/expansion bus bridge chip. Each master device is physically connected to the arbiter via a separate pair of lines, with each of them being used as REQ# (request) signal or GNT# (grant) signal, respectively. Ideally, the bus arbiter should be programmable by the system. If it is, the startup configuration software can determine the priority to be assigned to each member by reading from the maximum latency (Max Lat) configuration register associated with each bus master (see Figure 5.11). The bus designer hardwires this register to indicate, in increments of 250 ns, how quickly the master requires access to the bus in order to achieve adequate performance.

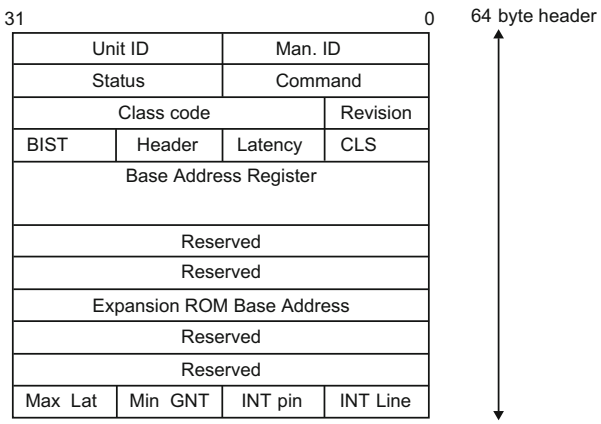


FIGURE 5.11

PCI configuration space.

At a given instant in time, one or more PCI bus master devices may require use of the PCI bus to perform a data transfer to another PCI device. Each requesting master asserts its REQ# output to confirm to the bus arbiter its pending request for the use of the bus. In order to grant the PCI bus to a bus master, the arbiter asserts the device's respective GNT# signal. This grants the bus to a bus master for one transaction, as shown in Figure 5.10. If a master generates a request, it is subsequently granted the bus and does not then initiate a transaction by asserting FRAME# signal within 16 PCI clocks after the bus goes idle, the arbiter may then assume that this bus master is malfunctioning. The action taken by the arbiter would then depend upon the system design. If a bus master has another transaction to perform immediately after the one it just initiated, it should keep its REQ# line asserted when it asserts the FRAME# signal to begin the current transaction. This informs the arbiter of its desire to maintain ownership of the bus after completion of the current transaction. In the event that ownership is not maintained, the master should keep its REQ# line asserted until it is successful in acquiring bus ownership again.

At a given instant in time, only one bus master may use the bus. This means that no more than one GNT# line will be asserted by the arbiter during any PCI clock cycle. On the other hand, a master must only assert its REQ# output to signal a current need for the bus. This means that a master must not use its REQ# line to "park" the bus on itself. If a system designer implements a bus parking scheme, the bus arbiter design should indicate a default bus owner by asserting the device's GNT# signal when no request from any bus masters are currently pending. In this manner, signal REQ# from the default master is granted immediately once no other bus masters require the use of the PCI bus.

The PCI specification does not define the scheme to be used by the PCI bus arbiter to decide the winner of any competition for bus ownership. The arbiter may utilize any scheme, such as one based on fixed, or rotational priority, or a combination of these two, to avoid deadlocks. However, the central arbiter is required to implement a fairness algorithm to avoid deadlocks. Fairness means that each potential bus master must be granted access to the bus independently of other requests. Fairness is defined as a policy that ensures that high-priority masters will not dominate the bus to the exclusion of lower-priority masters when they are continually requesting the bus. However, this does not mean that all agents are required to have equal access to the bus. By requiring a fairness algorithm there are no special conditions to handle when the signal LOCK# is active (assuming a resource lock) or when cacheable memory is located on the PCI. A system that uses a fairness algorithm is still considered fair if it implements a complete bus lock instead of a resource lock. However, the arbiter must advance to a new agent if the initial transaction attempting to establish a lock is terminated with retry.

(3) Interrupt routing

The host/PCI bus bridge will transfer the interrupt acknowledgment cycle from the processor to the PCI bus, which requires that the microprocessor chipset has an interrupt routing functionality. This router for the interrupt routing could be implemented using an Intel APIC I/O module, as given in Figure 5.3(a). This module can be programmed to assign a separate interrupt vector (interrupt table entry number) for each of the PCI interrupt request lines. It can also be programmed so that it realizes that one of its inputs is connected to an Intel programmable interrupt controller. If a system does not have this kind of controller, the microprocessor chipset should incorporate a software programmable interrupt routine device. In this case, the startup configuration software of the microprocessor attempts to program the router to distribute the PCI interrupt in an optimal fashion.

Whenever any of the PCI interrupt request lines is asserted, the APIC I/O module supplies the vector (see Figure 5.5 for an interrupt vector table) associated with that input to the processor's embedded local APIC I/O module. Whenever this programmable interrupt controller generates a request, the APIC I/O informs the processor that it must poll this programmable interrupt controller to get this vector. In response, the Intel processor can generate two back-to-back Interrupt Acknowledge transactions. The first Interrupt Acknowledge forces this programmable interrupt controller to prioritize the interrupts pending, while the second Interrupt Acknowledge requests that the interrupt controller send the vector to the processor. For a detailed discussion of APIC operation, refer to the MindShare book entitled Pentium Processor System Architecture (published by Addison-Wesley). For a detailed description of the Programmable Interrupt Controller chipset, refer to section 6.3 of this textbook.

Figure 5.10 can also be used to explain an interrupt acknowledgment cycle on the PCI bus, where a single byte enable is asserted. The PCI bus performs only one interrupt acknowledgment cycle per interrupt. Only one device may respond to the interrupt acknowledgment; that device must assert DEVSEL#, indicating that it is claiming the interrupt acknowledgment. The sequence is as follows:

1. During the address phase, the AD signals do not contain a valid address; they must be driven with stable data so that parity can be checked. The C/BE# signals contain the interrupt acknowledge command code (not shown).
2. IRDY# and the BE#s are driven by the host/PCI bus bridge to indicate that the bridge (master) is ready for response.
3. The target will drive DEVSEL# and TRDY# along with the vector on the data bus (not shown).

(4) Configuration registers

Each PCI device has 256 bytes of configuration data, which is arranged as 64 registers of 32 bits. It contains a 64-byte predefined header followed by an extra 192 bytes which contain extra configuration data. Figure 5.11 shows the arrangement of the header. The definitions of the fields in this header are as follows:

1. Unit ID and Man. ID. A Unit ID of FFFFh defines that there is no unit installed, while any other address defines its ID. The PCI SIG, which is the governing body for the PCI specification, allocates a Man. ID. This ID is normally shown at BIOS start-up.
2. Status and command.
3. Class code and revision. The class code defines the PCI device type. It splits into two 8-bit values with a further 8-bit value that defines the programming interface for the unit. The first defines the unit classification, followed by a subcode which defines the actual type.

(1) BIST, header, latency, CLS

The built-in-self test (BIST) is an 8-bit field, where the most significant bit defines whether the device can carry out a BIST, the next bit defines whether a BIST is to be performed (a 1 in this position indicates that it should be performed), and bits 3 0 define the status code after the BIST has been performed (a value of zero indicates no error). The header field defines the layout of the 48 bytes after the standard 16-byte header. The most significant bit of the header field defines whether the device is a multifunction device or not. A1 defines a multifunction unit. The cache line size (CLS) field defines

the size of the cache in units of 32 bytes. Latency indicates the length of time for a PCI bus operation, where the amount of time is the latency + 8 PCI clock cycles.

(2) Base address register

This area of memory allows the device to be programmed with an I/O or memory address area. It can contain a number of 32- or 64-bit addresses. The format of a memory address is (i) Bit 64-4: base address; (ii) Bit 3: PRF. Prefetching, 0 identifies not possible, 1 identifies possible; (iii) Bit 2, 1: Type. 00, any 32-bit address; 01, less than 1 MB; 10, any 64-bit address; and 11, reserved; (iv) Bit 0: 0. Always set a 0 for a memory address. For an I/O address space it is defined as: (i) Bit 31-2: base address; (ii) Bit 1, 0: 01. Always set to a 01 for an I/O address.

(3) Expansion ROM base address

This allows a ROM expansion to be placed at any position in the 32-bit memory address area.

(4) Max Lat, Min GNT, INT-pin, INT-line

The Min GNT and Max Lat registers are read-only registers that define minimum and maximum latency values. The INT-line field is a 4-bit field that defines the interrupt line used (IRQ0 IRQ15). A value of 0 corresponds to IRQ0 and a value of 15 corresponds to IRQ15. The PCI bridge can then redirect this interrupt to the correct IRQ line. The 4-bit INT pin defines the interrupt line that the device is using. A value of 0 defines no interrupt line, 1 defines INTA, 2 defines INTB, and so on.

5.2 MULTICORE MICROPROCESSOR UNITS

Multicore processors represent a major evolution in computing technology that beyan at the beyinning of the 21st century. It came at a time when businesses and industries re beginning to exploit the exponential growth of digital data and the globalization of the Internet. Multicore processors will eventually become the pervasive model in computing, business, communication, control and automation, and entertainment due to theiy performance and productivity benefits beyond the capabilities of single-core processors.

5.2.1 Introduction and basics

A multicore processor (also called a chip-level or on-chip multiprocessor) can be defined as a processor with more than one physical core on a single chip (or "die"). Each core contains its own dedicated processing resources similar to an individual CPU, except for the processor side bus, which may be shared between cores. Each core implements optimizations such as superscalar execution, pipelining, and multithreading independently. A system with *n* cores is most effective when it is presented with *n* or more threads concurrently. For example, a dual-core processor contains two cores, and a quad-core processor contains four cores. Therefore, for our purposes, a traditional microprocessor can be called a single-core processor.

(1) From single core to multicore

From the introduction of Intel's 8086 through to the Pentium 4, an increase in performance, has been seen with each increase in processor frequency. For example, the Pentium 4 ranged in speed

(frequency) from 1.3 to 3.8 GHz over its 8-year lifetime. The physical size of chips has decreased as the number of transistors per chip has increased; clock speeds increased, which boosted the heat buildup across the chip to a dangerous level. Intel's co-founder, Gordon Moore, is credited with Moore's law, which has been one of the guiding principles of computer architecture for more than two decades. In 1965, he stated that the number of transistors on a chip will roughly double each year (he later refined this, in 1975, to every two years).

Many techniques are used. performance within a single-core processor chip to Superscalar processors with the ability to issue multiple instructions concurrently are the standard. In these pipelines, instructions are prefetched, split into subcomponents and executed out-of-order. A major focus of computer architectures is the branch instruction. In order to speed up this process, the processor predicts which path will be taken; if the wrong path is chosen the processor must throw out any data computed while taking the wrong path and backtrack to take the correct path. Often even when an incorrect branch is taken the effect is equivalent to having waited to take the correct path. Branches are also removed by using loop unrolling, and sophisticated neural network-based predictors are used to minimize the misprediction rate. Other techniques used for performance enhancement include register renaming, trace caches, reorder buffers, dynamic software scheduling, and data value prediction

Speeding up processor frequency had run its course in the earlier part of the last decade; computer architects needed a new approach to improve performance. Adding an additional processing core to the same chip would, in theory, result in twice the performance and produce less heat; though in practice the actual speed of each core is slower than the fastest single-core processor. Due to advances in integrated circuit technology, multicore processor technology has become a mainstream element of CPU design, with Intel, AMD, IBM, SUN and Azul etc. all introducing many commercially available multicore chips.

The story for multicore processors started in 2001 and can be summarized as follows:

1. The first dual-core processor in the world, POWER4, was released in 2001 by IBM.
2. AMD promised its first quad-core Opteron processors by midyear in 2005. They would be manufactured using a "native" design that placed four independent cores on a single chip.
3. In late 2006 Intel introduced its first quad-core Xeon processors. The processors were manufactured by packaging two Intel dual-core processors in a single chip. Intel planned to deliver a dual-core desktop processor based on its 65-nm process technology in 2006.
4. IBM has been offering quad-core Power processors since 2005. Similar to Intel, the chips are manufactured using a multichip module.
5. SUN introduced the Sparc-based Niagara in late 2005. The chip has eight cores, each operating with four independent threads. By midyear, SUN promises Niagara 2, which will have eight cores, each with eight threads.
6. Azul Systems Inc. introduced its 24-core Vega chip in 2005. In December, the company announced Vega 2, a 48-core processor.

As with any technology, multicore architectures from different manufacturers vary greatly. Intel and AMD are the mainstream manufacturers of microprocessors. Intel produces many different flavors of multicore processors; the Pentium D in desktops, Core 2 Duo is used in both laptop and desktop environments, and the Xeon processor is used in servers. AMD has the Althon lineup for

desktops, Turion for laptops, and Opteron for servers/workstations. Although the Core 2 Duo and Athlon 64 X2 run on the same platforms, their architectures differ greatly.

Along with differences in communication and memory configuration, another difference is due to many cores the microprocessor has, and the functions of the different cores. Intel's current 7300 chipset-based platform combined with the Quad-Core Xeon 7300 processor is the industry's virtualization platform of choice for microprocessor servers from the second half of 2008. Intel's traditional rival, AMD, is portrayed by the media as the company with the winning strategy, and 64-bit computing and multicore technology are touted as two areas where AMD has a competitive edge over Intel. Intel has recently bridged the gap with AMD in 64-bit computing with its latest impressive offerings in multicore technology.

In contrast to commercially available two- and four-core machines in 2008, some experts believe that "by 2017 embedded processors could sport 4,096 cores, server CPUs might have 512 cores and desktop chips could use 128 cores". This rate of growth is astounding, considering that current desktop chips are only on the cusp of using four cores, and a single-core processor has been used for the past 40 years.

(2) Multicore challenges and open problems

Having multiple cores on a single chip gives rise to some problems and challenges. Power and temperature management are two concerns that can increase exponentially with the addition of multiple cores. Cache coherence is another challenge, since all designs discussed above have distributed L1 and in some cases L2 caches which must be coordinated. And finally, using a multicore processor to its full potential is another issue. If programmers do not write applications that take advantage of multiple cores there is no gain, and in some cases there is a loss of performance. Applications need to be written so that different parts can be run concurrently (without any ties to another part of the application that is being run simultaneously).

(a) Power and temperature

If two cores were placed on a single chip without any modification, the chip would, in theory, consume twice as much power and generate a large amount of heat. In the extreme, if a processor overheats the computer may even combust. To account for this, each design runs multiple cores at a lower frequency to reduce power consumption, and to combat unnecessary power consumption, many designs also incorporate a power control unit that has the authority to shut down unused cores or limit the amount of power used. By powering-off unused cores, and using clock gating, the amount of leakage in the chip is reduced. To lessen the heat generated by multiple cores on a single chip, the chip is designed to minimize the number of hot spots and sothat the heat is spread out across the chip. The processor follows a common trend to build temperature monitoring into the system, with its one linear sensor and ten internal digital sensors.

(b) Cache coherence

Cache coherence is a concern in a multicore environment because of distributed L1 and L2 caches. Since each core has its own cache, the copy of the data in that cache may not always be the most up-to-date version. For example, imagine a dual-core processor where each core brought a block of memory into its private cache, and then one core writes a value to a specific location. When the second core

attempts to read that value from its cache, it will not have the most recent version unless its cache entry is invalidated and a cache miss occurs. This cache miss forces the second core's cache entry to be updated. If this coherence policy was not in place, the wrong data would be read and invalid results would be produced, possibly crashing the program or the entire computer.

In general there are two schemes for cache coherence; a snooping protocol and a directory-based protocol. The snooping protocol only works with a bus-based system, and uses a number of states to determine whether or not it needs to update cache entries, and whether it has control over writing to the block. The directory-based protocol can be used on an arbitrary network and is, therefore, scalable to many processors or cores. This contrast swith snooping, which is not scalable. In this scheme, a directory is used which holds information about which memory locations are being shared in multiple caches, and which are used exclusively by one core's cache. The directory knows when a block needs to be updated or invalidated.

Intel's Core 2 Duo tries to speed up cache coherence by being able to query the second core's L1 cache and the shared L2 cache simultaneously. Having a shared L2 cache also has the added benefit that a coherence protocol does not need to be set for this level. AMD's Athlon 64 X2, however, has to monitor cache coherence in both L1 and L2 caches. This is made fasterby using the Hyper Transport connection, but still has more overhead than Intel's model.

Extra memory will be useless if the amount of time required for memory requests to be processed does not improve as well. Redesigning the interconnection network between cores is acrrently a major focus of chip manufacturers. A faster network means a lower latency in inter-core communication and memory transactions. Intel is developing their Quick path interconnect, which is a 20-bit wide bus running between 4.8 and 6.4 GHz; AMD's new Hyper Transport 3.0 is a 32-bit wide bus and runs at 5.2 GHz. Using five mesh networks gives the Tile architecture a per core bandwidth of up to 1.28 Tbps (terabits per second).

Despite these efforts, the question remains; which type of interconnect is best suited for multicore processors? Is a bus-based approach better than an interconnection network? Or is a hybrid like the mesh network a better optica?

(c) Multithreading

The last, and most important, issue is using multithreading or other parallel processing techniques to get the most performance out of the multicore processor. Rebuilding applications to be multithreaded means a complete rework by programmers to take best advantage of multicore systems. Programmers have to write applications with subroutines which can run in different cores, meaning that data dependencies will have to be resolved or accounted for (e.g. latency in communication or using a shared cache). Applications should be balanced. If one core is being used much more than another, the programmer is not taking full advantage of the multicore system. Some companies have designed new products with multicore capabilities; Microsoft and Apple's newest operating systems can run on up to four cores, for example.

Developing software for multicore processors raises some concerns. How does a programmer ensure that a high-priority task gets priority across the whole processor, not just one core? In theory, even if a thread had the highest priority within the core on which it is running, it might not have a high priority across the system as a whole. Another necessary tool for developers is debugging. However, how do we guarantee that the entire system stops and not just the core on which an application is running?

These issues need to be addressed along with the teaching of good parallel programming practices for developers. Once programmers have a basic grasp on how to multithread and program in parallel, instead of sequentially, ramping up to follow Moore's law will be easier.

5.2.2 Types and architectures

The multicore processor is a special kind of a multiprocessor in which all processors are resident on the same chip. There are two ways of differentiating multicore processors from multiprocessors. Firstly, a multicore processor is a MIMD (multiple-instruction, multiple-data) multiprocessor. In a multicore, different cores execute different threads (multiple-instructions), operating on different parts of memory (multiple-data). Second, a multicore processor is a shared memory multiprocessor, in which all cores share the same memory.

Although the architectures of multicore processors are much more complex than those of single-core processors, there are some common features between them. Figure 5.12 sketches the difference between their architectures. For a single-core processor, one CPU chip just has one core, composed of an ALU (arithmetic logic unit), memory registers and a bus interface with the system bus. However, a multicore chip can have more than one core, although each of these cores also contains ALU, memory registers and bus interface.

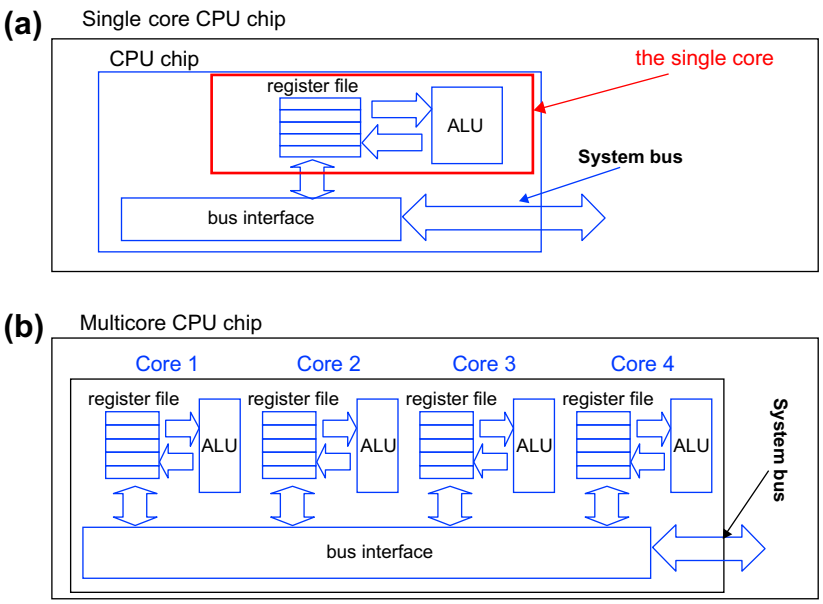


FIGURE 5.12

From single-core to multicore processor. (a) A single-core CPU chip; (b) a multicore CPU chip.

(1) Homogeneous and heterogeneous multicore processors

There are two ways of implementing multicore processors. One is homogeneous, or symmetric chip multiprocessing. This involves geometrically increasing the number of cores with each advance in feature size, so either duplicating or quadrupling the same core, and interconnecting them to produce a single, more powerful core as shown in Figure 5.13. We can see such an architecture in the Niagara chip and in the Quad-SHARC DSP, where four similar cores are connected to each other by a crossbar switch, as shown in Figure 5.14.

The other way is heterogeneous or asymmetric chip multiprocessing. This includes a combination of a large core, plus several low-power cores, such as a CPU surrounded by different specialized cores like GPUs (graphical processing units) and others. A heterogeneous, or asymmetric, multicore processor is one does have multiple cores on a single chip, but those cores might be of different designs of both high and low complexity. Figure 5.13 also shows a sample floorplan of an asymmetric chip multiprocessor. It consists of one high-performance core on the left top, and three medium-performance cores on the right and lower left. Of course, a real heterogeneous multicore can be much more complicated in floorplan. Up to now, heterogeneous multicore processors exist in two typical configurations; Cell multiprocessors and SoC (system on chip), which are discussed in the following paragraphs.

Both approaches will to benefit the industry on the whole. Both are aimed at pursuing a similar goal; increased throughput through parallel hardware, which requires changes to existing software to take advantage of this increased computing power. Many server applications focus primarily on throughput per unit cost and unit power. Desktop users are more interested in single applications or

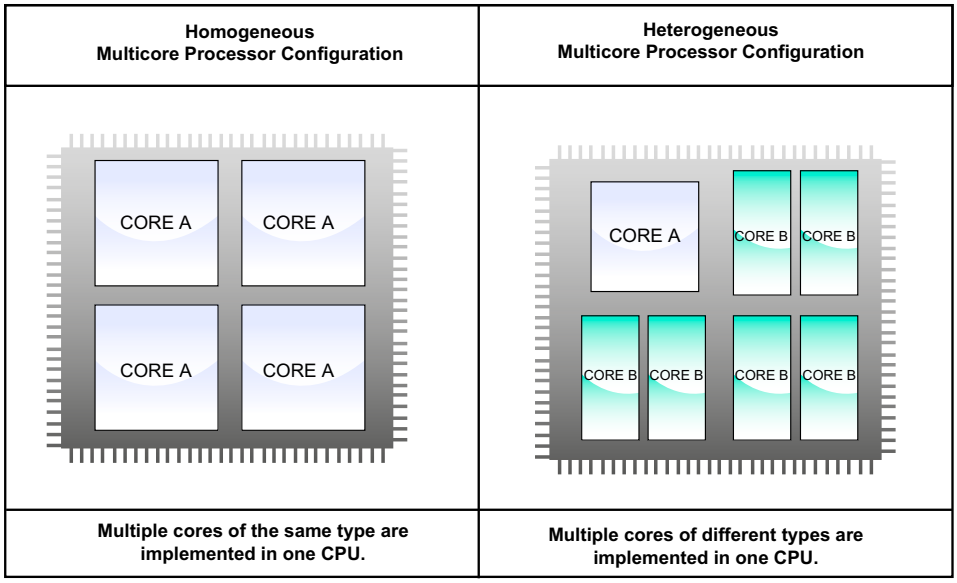
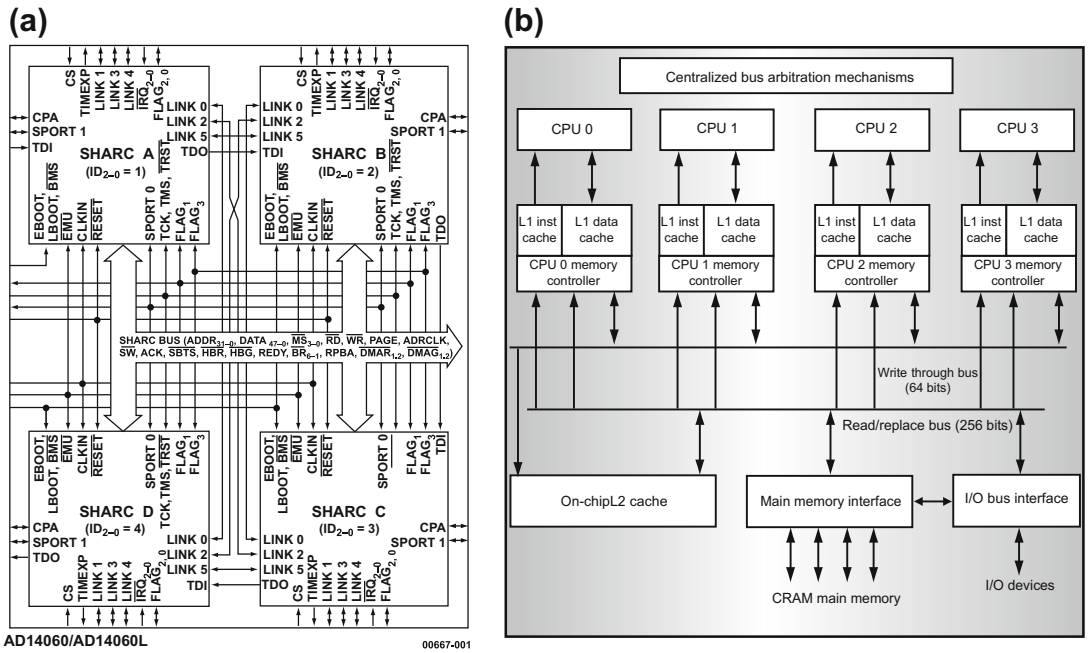


FIGURE 5.13

A sketch diagram for both homogeneous (symmetric) and heterogeneous (asymmetric) multicore processors.



Quad-SHARC Function Diagram

Quad-SHARC Block Diagram

FIGURE 5.14

An example of homogeneous multicore processors. (a) Quad-SHARC function diagram; (b) Quad-SHARC block diagram.

a few applications at the same time. A multicore processor for desktop users will probably be focused on a smaller number of larger, yet powerful, cores with better single-thread performance.

As the processor becomes more powerful and complex, both its peak and average power increase dramatically. The ability to dynamically switch between different cores and power-down unused cores is the key in asymmetric chip (multicore) multiprocessing. It has been shown that a representative heterogeneous processor using two core types achieves as much as a 63 percent performance improvement over an equivalent-area homogeneous processor. Asymmetric multiprocessors achieve better coverage of a spectrum of load levels.

Using a heterogeneous multicore configuration could significantly reduce processor power dissipation problems. Power consumption and heat dissipation have become key challenges in high-performance processor designs. Recently, the industry has introduced techniques for power reduction, such as clock gating and voltage/frequency scaling, both of which are very effective when applied at the single-core level.

For multiple cores, however, these techniques have limitations. Data must be passed onto different parts of the processor, and may be passed on unused portions that have been gated off, which consumes a substantial amount of power. Gating only reduces dynamic power. Large unused portions of the chip still consume leakage power. This is the same with voltage and frequency scaling techniques.

We believe the best choice core of complex is a heterogeneous multicore processor with both high- and low-complexity cores. Recent research in heterogeneous or asymmetric multicore processors has identified that they have significant advantages over homogeneous multicore processors in terms of power and throughput, and in the performance of parallel applications.

(2) Cell processor

As explained in the previous paragraphs, the "Cell processor" is one of the possible heterogeneous configurations of multicore processors. A Cell processor consists of both control-intensive processor and compute-intensive processor cores, each with its own distinguishing features. This section will discuss its physical architecture.

The Cell architecture includes software and hardware cells. A software cell is a program plus its associated data, and a hardware cell is an execution unit with the capability to execute a software cell. There is no fixed architecture (the software cells could float around and can be processed by any available hardware cell), which is very interesting, when combined with the ubiquitous computing idea. Several of these hardware cells can create a bigger Cell computer (Ad Hoc). But this is only an architectural vision; a real-life Cell processor is bound to a single chip, and floating Cells are still only ideas. But the general principle remains the same. A Cell processor consists of a PPE (PowerPC processing element), which acts as the main processor to distribute tasks (software cells). It also has a MFC (memory flow controller), which interfaces between the computing and memory units, and also many SPE (synergistic processing elements), which are hardware cells having their own memory. Nowadays, the Cell CPU is essentially a combination of a power processor with eight small vector processors. All these units are connected via an EIB (element interconnect bus) and communicate with peripheral devices or other CPUs via the FlexIO interface. A single Cell, essentially a network on chip, offers up to 256 Gflop, single precision, floating-point performance. A block diagram of the Cell processor is shown in Figure 5.15.

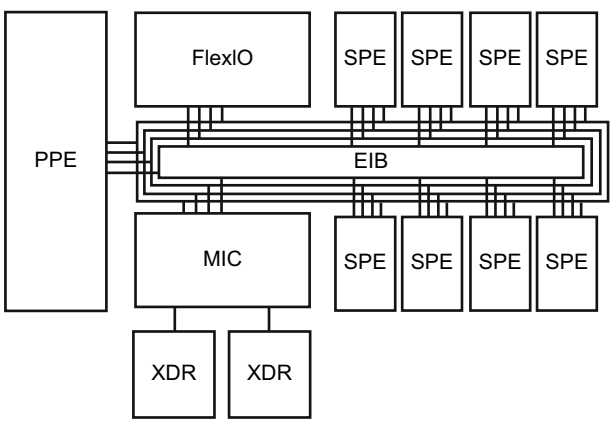


FIGURE 5.15

Cell processor architecture.

(a) Prototype

A prototype was produced by using 90 nm silicon on insulator technology, with eight copper layers (wiring). It consists of 241 million transistors at 235 per square mm, and consumes 60 80 W. IBM's virtualization technology is incorporated in the PPE. The CPU is clocked at 4 GHz at 1.1 V.

(b) The power processing element

The power processing element (PPE) offers the normal PowerPC ISA (instruction set architecture). It is a dual-threaded 64-bit power processor. Its architecture is very simple to guarantee high clock rates, so it uses only in-order execution with a deep super scalar 2-way pipeline with more than 20 stages. It offers a 2×32 kB L1 split cache, a 512 kB L2 cache and virtualization. Altogether the PPE can be seen as a simplified Power processor.

(c) The synergistic processing element (SPE)

The SPE is essentially a full blown vector CPU with its own RAM (memory). Its ISA has a fixed length of 32 bits. Current SPEs have about 21 million transistors, of which two thirds are dedicated to the SRAM (memory). The processor has no branch prediction or scheduling logic, but relies on the programmer/compiler to find parallelism in the code. It uses two independent pipelines, and issues two instructions per cycle; one SIMD (single-instruction, multiple-data) computation operation and one memory access operation. All instructions are processed strictly in order, and each instruction works with 128-bit compound data items. Four, single-precision, floating-point units and four integer units offer up to 32 GOps each. The single-precision, floating-point, units are not IEEE-754-compliant in terms of rounding and special values, and they can also be used to compute double-precision-floating point numbers which are compliant to the IEEE-754 standard. Their computation is, however, rather slow (3-4 Gflops). The schematic architecture of a single SPE is shown in Figure 5.16. The memory layout of the SPE is also quite special; each SPE has its own 256 kB RAM, which is called local storage (LS). This SRAM storage can be accessed extremely fast in 128-bit lines. Additionally, each SPE has a large register file of 128×128-bit registers which store all available data types. There is no cache, virtual memory support or coherency for the local storage and the data can be moved with DMA

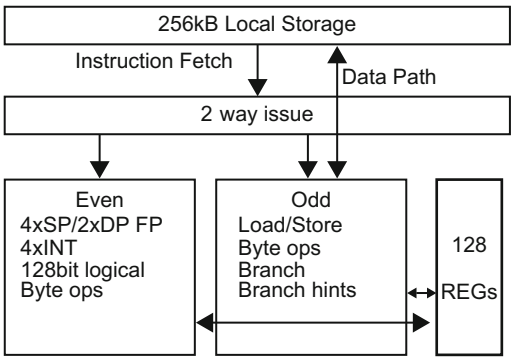


FIGURE 5.16

The SPE (synergistic processing element) architecture in Cell processors.

(direct memory access) from/to main memory via the EIB (element interconnect bus). The memory flow controller (MFC) acts like a MMU in this context and provides virtual memory translations for main memory access.

(d) The element interconnect bus

The EIB is the central communication channel inside a Cell processor; it consists of four 128-bit-wide concentric rings. The ring uses buffered, point to point communication to transfer the data and so is scalable. It can move 96 bytes per cycle and is optimized for 1024-bit data blocks. Additional nodes (e.g. SPEs) can be added easily and increase only the maximal latency of the ring. Each device has a hardware guaranteed bandwidth of 1/numDevices to enhance its suitability for real-time computing.

(e) The I/O interconnect FlexIO

The I/O interconnect connects the Cell processor (the EIB) to the external world (e.g. other Cell processors). It offers 12 unidirectional byte-lanes which are 96 wires. Each lane may transport up to 6.4 GB/s, which makes 76.8 GB accumulated bandwidth. Seven lanes are outgoing (44.8 GB/s) and five lanes incoming (32 GB/s). There are cache coherent (CPU interconnects) and noncoherent links (device interconnects) and two Cell processors can be connected loosely.

(f) The memory interface controller (MIC)

The MIC connects the EIB to the main DRAM memory, which is in this case Rambus XDR memory, which offers a bandwidth of 25.2 GB/s. The memory shows that the Cell will be used for more than game consoles and consumer electronics. The MIC offers virtual memory translation to the PPE and the SPEs. The memory itself is not cached; only the PPE has its own cache hierarchy.

A variety of Cell programming models have been suggested sofar. The most basic among them is shown in Figure 5.17, where the PPE is used for execution of the main program and the SPEs for execution of a sub-program. The main program executed on the PPE (hereinafter called the PPE program) divides a sequence of processes or data, and distributes a segment to every SPE for processing by the sub-program running on the SPEs (hereinafter called the SPE program). Upon completion of the requested operation, the SPE program returns the processed result to the PPE program.

Let us take a look at how PPE and SPE programs are executed on the Cell architecture, together with how necessary data are transmitted and received. As explained in Figure 5.18, the dataflow consists of the steps given below:

1. (PPE Program) Loads the SPE program to the LS.
2. (PPE Program) Instructs the SPEs to execute the SPE program.
3. (SPE Program) Transfers required data from the main memory to the LS.
4. (SPE Program) Processes the received data in accordance with the requirements.
5. (SPE Program) Transfers the processed result from the LS to the main memory.
6. (SPE Program) Notifies the PPE program of the termination of processing.

(3) Intel and AMD multicore architectures

Intel and AMD are the mainstream manufacturers of microprocessors. Intel produces many different flavors of multicore processors; the Pentium D is used in desktops, Core 2 Duo is used in both laptop

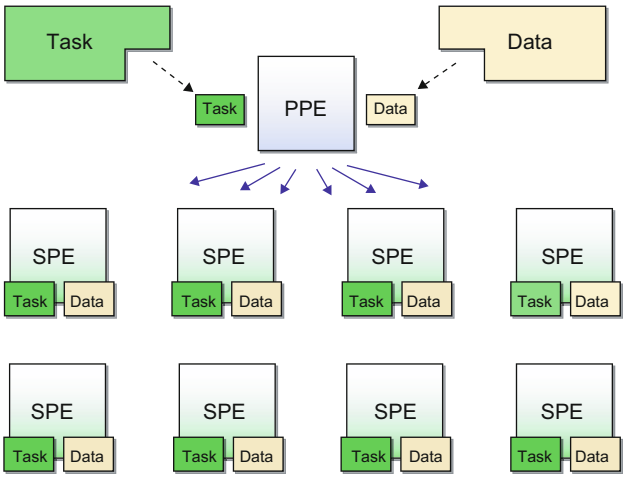


FIGURE 5.17

Programming model of a basic Cell processor.

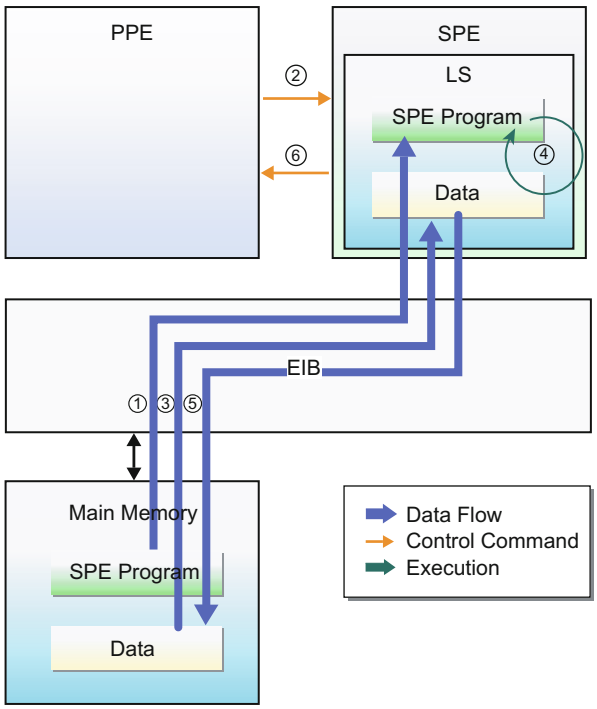


FIGURE 5.18

The program control and dataflow in the basic programming model for the Cell processor.

and desktop, and the Xeon processor is used in servers. AMD has the Althon for desktops, Turion for laptops, and Opteron for servers and workstations. Although the Core 2 Duo and Athlon 64 X2 run on the same platforms, their architectures differ greatly. Figure 5.19 shows block diagrams for the Core 2 Duo and Athlon 64 X2. Both architectures are homogeneous dual-core processors. The Core 2 Duo adheres to a shared memory model, with private L1 caches and a shared L2 cache. If a L1 cache miss occurs, both the L2 cache and the second core's L1 cache are traversed in parallel before sending a request to main memory. By contrast, the Athlon follows a distributed memory model with discrete L2 caches, which share a system request interface, eliminating the need for a bus.

AMD engineers invented Hyper-Transport technology, which presents a high-bandwidth (10 GB/s) low-latency (less than 10 ns) chip-to-chip interconnect mechanism. After inventing this technology, AMD worked to share it with the industry. AMD has also created an enhanced version of Hyper-Transport technology, known as "Coherent Hyper-Transport technology", that it uses to coordinate the contents of on-chip caches in multiprocessor AMD64 configurations. Figure 5.20 explains these larger configurations, which are best suited for executing computationally intensive applications and for database management. The system request interface also connects the cores with an on-chip memory controller and the Hyper-Transport interconnection. Hyper-Transport effectively reduces the number of buses required in a system, so reducing bottlenecks and increasing bandwidth.

The Larrabee architecture will be Intel's next step in evolving the visual computing platform. This includes a high-performance, wide SIMD vector processing unit along with a new set of vector instructions, including integer and floating-point arithmetic, vector memory operations and conditional

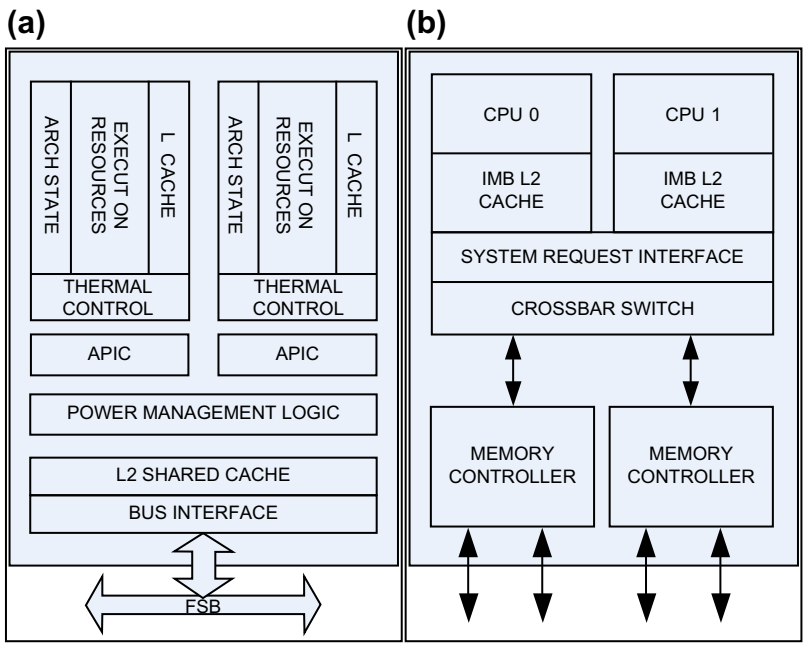


FIGURE 5.19

The architectures of two Intel and AMD multicore processors. (a) Intel Core 2 Duo; (b) AMD Athlon 64 X2.

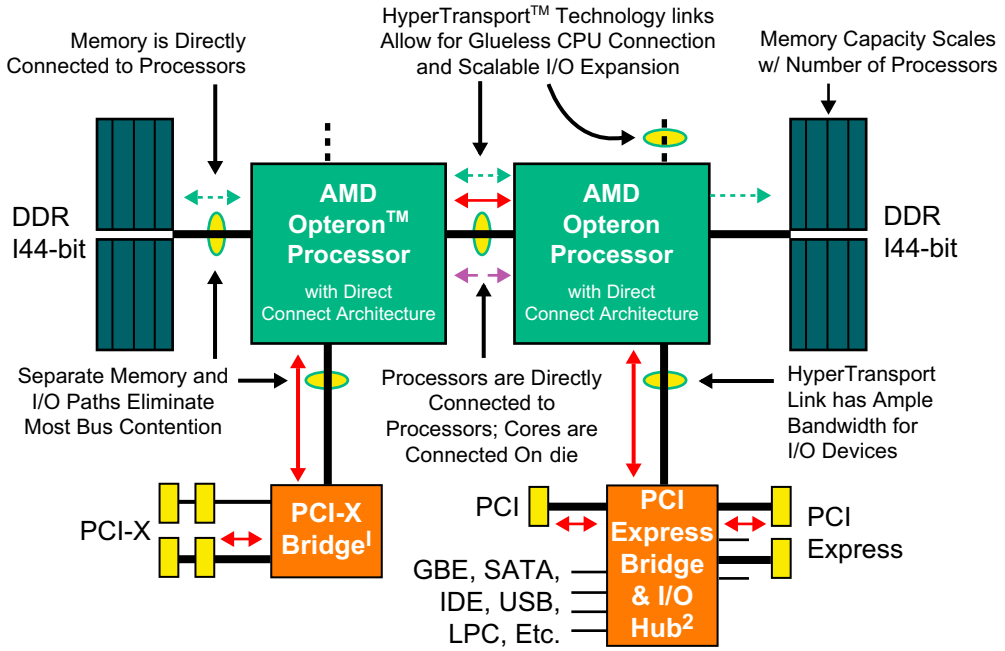


FIGURE 5.20

The Direct-Connect architecture with Hyper-Transport technology invented for the AMD multicore processors.

instructions. In addition, Larrabee includes a major, new, hardware coherent cache design enabling the many-core architecture. The architecture and instructions have been designed to deliver performance, energy efficiency and general-purpose programmability in order to meet the demands of visual computing and other workloads that are inherently parallel in their nature. Tools are critical to success, and key Intel software products will be enhanced to support the Larrabee architecture and enable unparalleled developer freedom. The industry APIs (application program interfaces), such as DirectX™ and OpenGL, will be supported on Larrabee-based products.

5.2.3 Cores and components

Similar to the single-core processor unit, the multicore processor unit works by relying on the processor and key components including cache, memory and bus. In this subsection, we will discuss these elements of multicore processors.

(1) Core microarchitecture

Some features of the new generations of Intel and AMD processors are brand new, but others are not. For example, the underlying architecture named x86 is still the same as the x86 CPU. This architecture defines which kind of instructions the processor can execute, and the way the processor can be accessed, i.e. which software can be used. Both Intel and AMD have been expanding the x86

instruction set continuously, for instance with special MMX (matrix math extension), SSE (streaming SIMD extension) and x86-64 instructions.

As another example, the Bobcat processor, made by AMD, is a very simplified x86 CPU core aiming at low-power x86 processing with values between 1 and 10 W, together with low-voltage operation. According to AMD roadmaps, the Bobcat processor will be incorporated together with GPU (graphic processing unit) cores into processors under the codename "Fusion" (AMD Fusion is the codename for a future microprocessor design, combining general processor execution as well as three-dimensional geometry processing and other functions of modern GPU into a single package.). New multicore processors now incorporate a variety of improvements, particularly in memory prefetching, speculative loads, SIMD execution and branch prediction, yielding an appreciable performance increase.

The defined microarchitecture is the hardware implementation of such architecture. So while AMD and Intel processors are based on the exact same x86 architecture and are therefore 100% compatible, their inner working is completely different due to different microarchitectures. Figure 5.21 explains the relation between architecture, microarchitecture and processors.

One new technology included in the design is Macro-Ops Fusion, which combines two x86 instructions into a single micro-operation. For example, a common code sequence such as a compare followed by a conditional jump would become a single micro-op. Other new technologies include one-cycle throughput (two cycles previously) of all 128-bit SSE instructions and a new power saving design. All components will run at minimum speed, ramping up speed dynamically as needed. This allows the chip to produce less heat, and consume as little power as possible.

(2) Cache memories

In multicore processors, the embedded cache memory architecture must be carefully considered because caches may be dedicated to a particular processor core, or shared among multiple cores. Furthermore, multicore processors typically employ a more complex interconnect mechanism, such as shared bus fabric, to connect the cores, caches, and external memory interfaces often including switches and routers.

In a multicore processor, cache coherency must also be considered. Multicore processors may also require that on-chip memory should be used as a temporary buffer to share data among multiple processors, as well as to store temporary thread context information in a multi-threaded system.

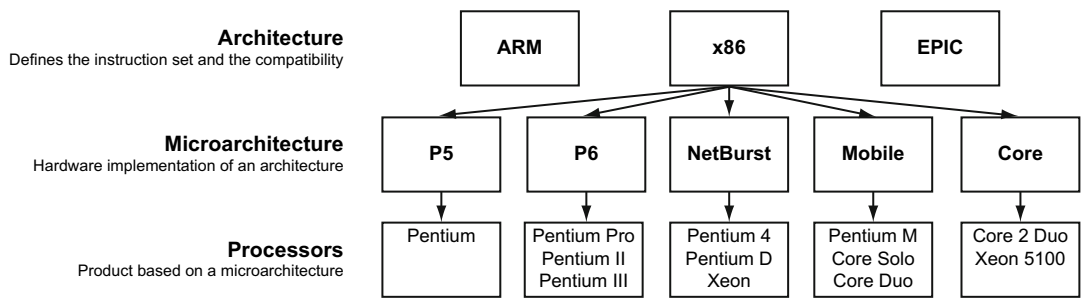


FIGURE 5.21

The relation between architecture, microarchitecture and processors in multicore processor units.

There are three main types of multicore processors as shown in Figure 5.22. (1) Simple multicore processors have two complete processors placed on the same die (chip) or package. (2) Shared-cache multicore processors consist of two complete cores, sharing some levels of cache, typically Level 2 (L2) or Level 3 (L3). (3) Multi-threaded multicore processors have multiple cores, with multiple threads within each core.

The following briefly introduces three of the main caching techniques for multicore processor chips; coherent cache, cooperative cache, and synergistic cache.

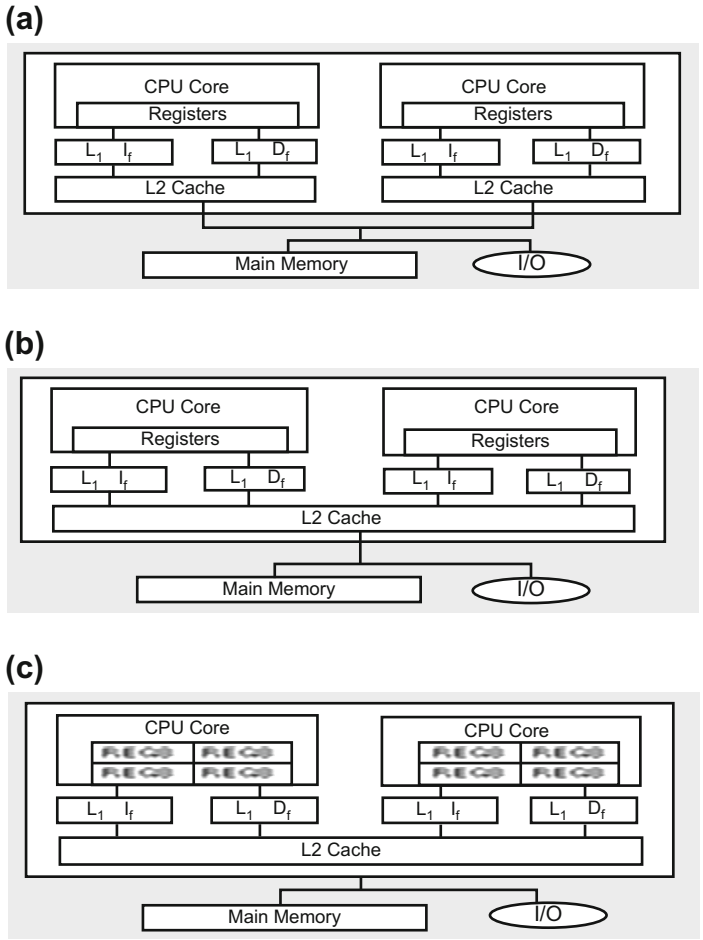


FIGURE 5.22

Three main types of multicore processors. (a) Simple multicore processors have two complete processors on the same die. (b) Shared-cache multicore processors consist of two complete cores, sharing some levels of cache, typically L2 or L3. (c) Multi-threaded multicore processors have multiple cores, with multiple threads within each core.

(a) Coherent caching technique

With the cache-based model, the only directly addressable storage is the off-chip memory. All on-chip storage is used for caches with hardware management of locality and communication. As cores perform memory accesses, the caches attempt to capture the application's working set by fetching or replacing data at the granularity of cache blocks. The cores communicate implicitly through loads and stores to the single memory image. Since many caches may store copies of a specific address, it is necessary to query multiple caches on load and store requests; and potentially invalidate entries to keep caches coherent. A coherence protocol, such as MESI, minimizes the cases when remote cache lookups are necessary. Remote lookups can be distributed through a broadcast mechanism, or by first consulting a directory structure. The cores synchronize by using atomic operations such as compare-and-swap. Cache-based systems must also provide event-ordering guarantees within and across cores following some consistency model. All issues including caching, coherence, synchronization, and consistency are implemented in hardware.

Coherent caching techniques were developed for board-level and cabinet-level systems, for which communication latency ranges from tens to hundreds of cycles. In multicore processor chips, coherence signals travel within one chip, where latency is much lower and bandwidth is much higher. Consequently, even algorithms with non-trivial amounts of communication and synchronization can scale reasonably well. Moreover, the efficient design points for coherent caching in multicore processor chips are likely to be different from those for the simple multicore processors and shared-cache multicore processors systems given in Figure 5.22.

(b) Cooperative caching technique

The basic concept of cooperative caching for multicore processor chips (dies) is inspired by software cooperative caching algorithms, which have been proposed and shown to be effective in the context of file and website caching. Although physically separate, individual file/website caches cooperatively share each other's space to meet their dynamic capacity requirements, thus reducing the traffic to the remote servers. In a similar vein, cooperative caching for multicore processor chips tries to create a globally managed, "shared", aggregate, on-chip cache via the cooperation of the different on-chip caches, so that the aggregate cache resources can better adapt to the dynamic caching demands of different applications.

We view cooperative caching as a unified framework for organizing the on-chip cache resources of a multicore processor chip. Two major applications for cooperative caching immediately come to mind; (1) optimizing the average latency of a memory request so as to improve performance, and (2) achieving a balance between dynamic sharing of cache resources and performance isolation, so as to achieve better performance and quality of service in a throughput-oriented environment.

We focus on the first application only. Cooperative caching tries to optimize the average latency of a memory request by combining the strengths of private and shared caches adaptively. This is achieved in three ways: (1) by using private caches as the baseline organization, it attracts data locally to reduce remote on-chip accesses, thus lowering the average on-chip memory latency; (2) through cooperation among private caches, it can form an aggregate cache which has similar effective capacity to a shared cache, to reduce costly off-chip misses; (3) by controlling the amount of cooperation, it can provide a spectrum of choices between the two extremes of private and shared caches, to better suit the dynamic workload behavior.

(c) Synergistic caching technique

We introduce synergistic caching to address the architecture challenge of designing a fast memory system while balancing memory bandwidth demand with chip area. Synergistic caching addresses these challenges by increasing effective memory performance and minimizing the area devoted to the caching system.

Caching was introduced to decrease average memory access time by using data manytimes by a single processor. Synergistic caching extends traditional caching to the multiprocessor environment by enabling multiple uses of data by multiple processors. It allows a processing node to retrieve data from a neighboring cache (through the on-chip network) instead of retrieving the data from main memory. When multiprocessors are added to the single-chip system, a processor has three locations from which to retrieve data: its local cache, main memory, or the neighboring cache.

(3) Shared bus fabric

A shared bus fabric (SBF) is a high-speed link that is needed to transfer data between processors, caches, I/O interfaces, and memories within a multicore processor system in a coherent fashion. It is the on-chip equivalent of the system bus for snoop-based shared-memory multiprocessors.

We model a MESI-like snoopy write invalidate protocol with write-back L2 caches for this discussion. The SBF needs to support several coherence transactions (request, snoop, response, data transfer, invalidates, etc.) as well as arbitrate access to the corresponding buses. Due to large transfer distances on the chip and high wire delays, all buses must be pipelined, and therefore unidirectional. Thus, these buses appear in pairs; typically, a request traverses from the requester to the end of one bus, where it is queued to be rerouted (possibly after some computation) across a broadcast bus that every node will eventually see, regardless of their position on the bus and distance from the origin. In the following discussion a bidirectional bus is really a combination of two unidirectional pipelined buses.

We are assuming, for this discussion, that all cores have private L1 and L2 caches, and that the shared bus fabric connects the L2 caches (along with other units on the chip and off-chip links) to satisfy memory requests and maintain coherence. Below we describe a typical transaction on the fabric.

(a) Typical transaction on the SBF

A load that misses in the L2 cache will enter the shared bus fabric to be serviced. First, the requester (in this case, one of the cores) will signal the central address arbiter that it has a request. Upon being granted access, it sends the request over an address bus (AB in Figure 5.23(a)). Requests are taken off the end of the address bus and placed in a snoop queue, awaiting access to the snoop bus (SB). Transactions placed on the snoop bus cause each snooping node to place a response on the response bus (RB). Logic and queues at the end of the response bus collect these responses and generate a broadcast message that goes back over the response bus identifying the action each involved party should take (e.g., source the data, change coherence state). Finally, the data are sent over a bidirectional data bus (DB) to the original requester. If there are multiple SBFs (e.g., connected by a P2P link), the address request will be broadcast to the other SBF via that link, and a combined response from the remote SBF returned to the local one, to be merged with the local responses. Note that the above transactions are quite standard for any shared memory multiprocessor implementing a snoopy write invalidate coherence protocol.

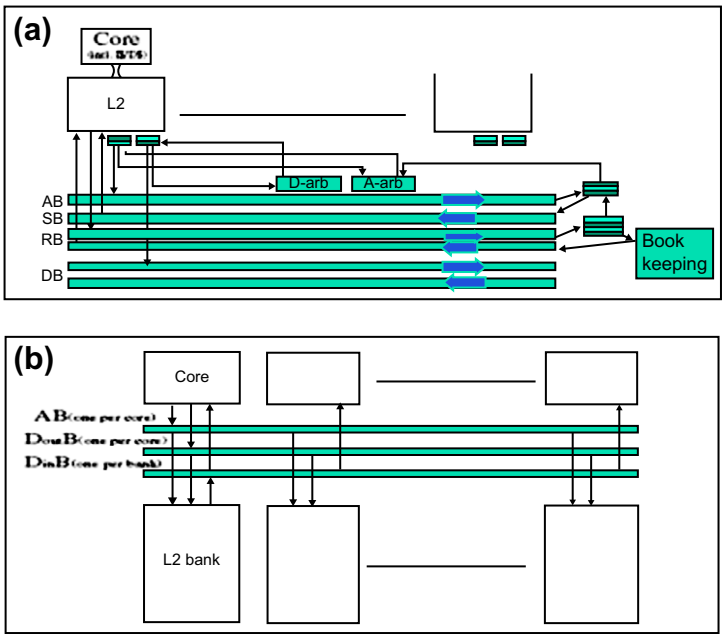


FIGURE 5.23

Shared bus fabric on a multicore processor chip. (a) The architecture of shared bus fabric; (b) a typical crossbar.

(b) Elements of the shared bus fabric

The composition of the SBF allows it to support all the coherence transactions mentioned above. We now discuss the primary buses, queues and logic that would typically be required for supporting these transactions. Figure 5.23(a) illustrates a typical SBF. Details of the modeled design, only some of which we have room to describe here, are based heavily on the shared bus fabric in the Power5 multicore architecture.

Each requester on the SBF interfaces with it via request and data queues. It takes at least one cycle to communicate information about the occupancy of the request queue to the requester. The request queue must therefore have at least two entries to maintain the throughput of one request every cycle. Similarly, all the units that can source data need to have data queues of at least two entries. Requesters connected to the SBF include cores, L2 and L3 caches, I/O devices, memory controllers, and non-cacheable instruction units.

All requesters interface to the fabric through an arbiter for the address bus. The minimum latency through the arbiter depends on (1) the physical distance from the central arbiter to the most distant unit, and (2) the levels of arbitration. Caches are typically given higher priority than other units, so arbitration can take multiple levels based on priority. Distance is determined by the actual floorplan. Since the address bus is pipelined, the arbiter must account for the location of a requester on the bus in determining what cycle access is granted. Overhead for the arbiter includes control signals to/from the requesters, arbitration logic and some latches.

After receiving a grant from the central arbiter, the requester unit puts the address on the address bus. Each address request goes over the address bus and is then copied into multiple queues, corresponding to outgoing P2P links (discussed later) and to off-chip links. Being a broadcast bus, the address bus spans the width of the chip. There is also a local snoop queue that queues up the requests and participates in the arbitration for the local snoop bus. Every queue in the fabric incurs at least one bus cycle of delay. The minimum size of each queue in the interconnection (there are typically queues associated with each bus) depends on the delay required for the arbiter to stall further address requests if the corresponding bus gets stalled. Thus it depends on the distance and communication protocol to the device or queue responsible for generating requests that are sinked in the queue, and the latency of requests already in transit on the bus. We therefore compute queue size based on floorplan and distance.

The snoop bus can be shared, for example by off-chip links and other SBF, so it also must be accessed via an arbiter, with associated delay and area overhead. Since the snoop queue is at one end of the address bus, the snoop bus must run in the opposite direction of the address bus, as shown in Figure 5.23(a). Each module connected to the snoop bus snoops the requests. Snooping involves comparing the request address with the address range allocated to that module (e.g., memory controllers) or checking the directory (tag array) for caches.

A response is generated after a predefined number of cycles by each snooper, and goes out over the response bus. The delay can be significant, because it can involve tag-array lookups by the caches, and we must account for possible conflicts with other accesses to the tag arrays. Logic at one end of the bidirectional response bus collects all responses and broadcasts a message to all nodes, directing their response to the access. This may involve sourcing the data, invalidating, changing coherence state, etc. Some responders can initiate a data transfer on a read request simultaneously with generating the snoop response, when the requested data are in an appropriate coherence state. The responses are collected in queues. All units that can source data to the fabric need to be equipped with a data queue. A central arbiter interfacing with the data queues is needed to grant one of the sources access to the bus at a time. Bidirectional data buses source data. They support two different data streams, one in either direction. Data bandwidth requirements are typically high.

It should be noted that designs are possible with fewer buses, and the various types of transactions multiplexed onto the same bus. However, that would require higher-bandwidth (e.g., wider) buses to support the same level of traffic at the same performance, so the overheads are unlikely to change significantly.

(c) P2P links

If there are multiple SBFs in the system, the connection between the SBFs is accomplished using P2P links. Multiple SBFs might be required to increase bandwidth, decrease signal latencies, or to ease floor planning (all connections to a single SBF must be on a line). Each P2P link should be capable of transferring all kinds of transactions (request/response/data) in both directions. Each P2P link is terminated with multiple queues at each end. There needs to be a queue and an arbiter for each kind of transaction described above.

(d) Crossbar interconnection system

The crossbar interconnection system consists of crossbar links and crossbar interface logic. A crossbar consists of address lines going from each core to all the banks (required for loads, stores,

prefetches), data lines going from each core to the banks (required for write backs) and data lines going from every bank to the cores (required for data reload as well as invalidate addresses). A typical implementation, shown in Figure 5.23(b), consists of one address bus per core from which all the banks feed. Each bank has one outgoing data bus from which all the cores feed. Similarly, corresponding to each write port of a core is an outgoing data bus that feeds all the banks.

5.2.4 Hardware implementation

As electronic and semiconductor technologies, in particular large-scale integrated circuit technologies, have developed, so the hardware implementation of multicore processor chips has become more and more advanced. With existing technology, all multicore processor chips can be designed with one of these architectures:

(a) The first is classical computer architecture with the system bus concept, in which much work has been done on bus architectures and arbitration strategies. In general, this work is tightly coupled with the memory architecture. The key problem with this approach upto is scaling incorporate a larger number of processors; if this is solved it becomes suitable for using system-on-chip (SoC) technology.

(b) The second approach is linked with networking, which resulted in the concept of networks-on-chip (NoC). Key NoC concepts include distributing the communication structure, and using multiple routes for data transfer. This produces flexible, programmable, and even reconfigurable networks. NoC allows targeting platforms to a much wider variety of products.

In this subsection, these two technologies, SoC and NoC, are briefly explained, with emphasis on their specifications for implementing multicore processor hardware. It must be emphasized here that both SoC and NoC for multicore processors are different from those for multiprocessors. As mentioned many times in this chapter, multicore processors are components that take advantage of increased transistor densities to put more processors on a single chip, but they do not try to leverage application needs. In contrast, multiprocessors are those custom architectures that balance the constraints of integrated circuit technology with an application's needs.

(1) System-on-chip for multicore processors

System-on-chip (SoC) is an integrated circuit that includes a processor, a bus, and other elements on a single monolithic substrate. Various components, such as volatile memory systems, non-volatile memory systems, data signal processing systems, I/O interface ASIC, mixed signal circuits and logic circuits, are each formed into units and integrated on a single chip. As technology advances, the integration of the various units included in a SoC design becomes increasingly complicated.

In the industry, a system-on-chip is designed by stitching together multiple stand-alone designs (cores) of large-scale integrated circuits to provide full functionality for an application.

(2) Network-on-chip for multicore processors

The network-on-chip (NoC) design paradigm is seen as a way of enabling the integration of an exceedingly high number of computational and storage blocks in a single chip. Although a complex SoC can be viewed as a micronetwork of multiple stand-alone blocks, models and techniques from networking, multiple-agent concurrency and parallel processing can be borrowed for the networking-oriented applications of multicore processors.

For NoC, the micronetwork must meet quality-of-service requirements (such as reliability, guaranteed bandwidth/latency), and deliver energy efficiency. It must do this under the limitation of intrinsically unreliable signal transmission media. Such limitations are due to the increased likelihood of timing and data errors, the variability of process parameters, crosstalk, and environmental factors such as electromagnetic interference and software errors.

To improve network-traffic performance, processor vendors have integrated multiple proprietary cores with integrated memory subsystems, called network processors. However, porting general-purpose processor software required for network services to a proprietary core with a limited amount of code space and proprietary development tools has relegated network processors to Layer 2 and Layer 3 types of network traffic processing. Adding further disarray to the above gap between communication processors and network processors is the fact that both processors fail to address the need for processing computationally intensive operations and network-services applications, such as encryption, transmission control protocol (TCP) offload, big volumes of data compression and decompression, look-ups, and regular-expression acceleration. This necessitates attaching separate coprocessors connected via additional system interfaces.

Figure 5.24 is a diagram of a typical multicore processor design using SoC/NoC technologies. In this figure, the cnMIPS cores represent a heterogeneic 2 16 of CPUs with 18 64 bits capability. In addition to those components including the shared L2 cache, packet I/O processor, etc., it can be integrated with some sockets such as PCI-X, TCP, MISC I/O, etc. With SoC/NoC technologies, all these can be incorporated on one single chip.

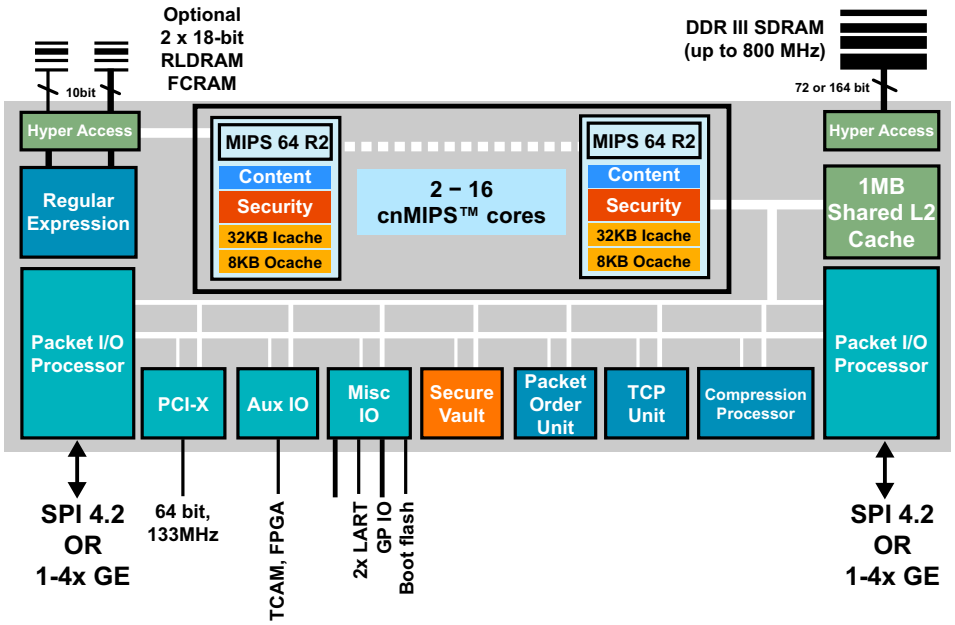


FIGURE 5.24

A schematic of typical multicore processor design with SoC/NoC technologies.

5.2.5 Software implementation

Getting software running on a multicore CPUs is, in many cases, fairly easy. Getting the software to make full use of all the processor cores simultaneously, obtaining parallelism, is, on the other hand, a real challenge. Parallelism is an important design consideration for image-processing systems, networking control planes, and other performance-hungry, computation-intensive applications. A key metric for such applications is system throughput, and the best way to maximize this is to either perform operations faster, or perform more operations concurrently (i.e. in parallel).

All operating systems vary in how they use multiple processor cores. Some support asymmetric multiprocessing (AMP), some support symmetric multiprocessing (SMP), (and some support bound multiprocessing (BMP), which is less popular). The key difference between these two models is simple; SMP supports parallel software execution, whereas AMP does not.

In the AMP model, each processor core has a separate operating system, or a separate copy of the same operating system. Each manages a portion of the system memory and a portion of system I/O resources. The memory layout and I/O partitioning is typically configured at system boot. In effect, a quad-core system appears as four separate systems as four processor chips in a multiple-chip system. Figure 5.25(a) is a diagrammatic explanation of the AMP model.

In the SMP model, a single instance of the operating system has access to all system memory and I/O resources. The operating system abstracts the number of CPU cores from the application software, allowing developers to create parallelized software that can scale as more cores are added. All

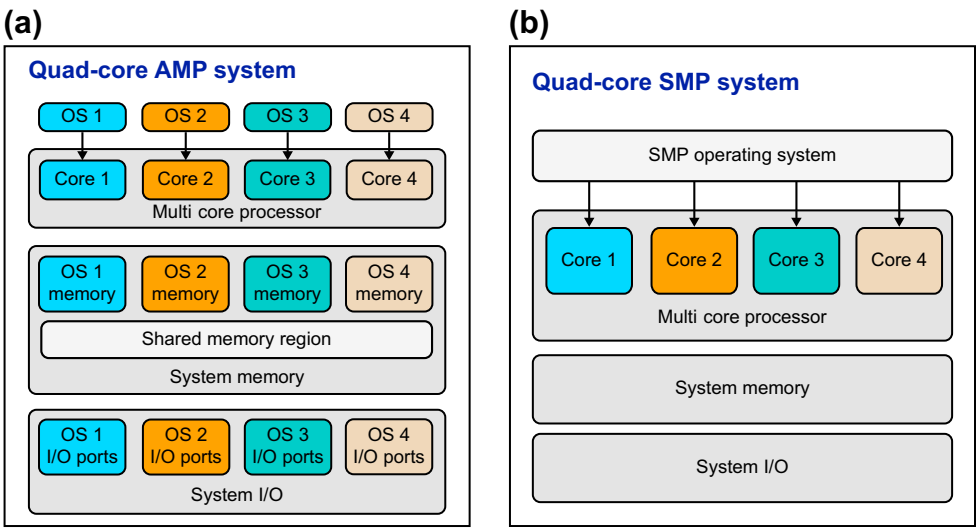


FIGURE 5.25

(a) In AMP, each core has a separate operating system that manages its own memory regions and I/O. The memory regions can overlap to provide a common shared memory region that allows applications on different cores to communicate with one another. (b) In the SMP model, a single instance of the operating system manages all processor cores. Applications have uniform access to memory and I/O, and can use the operating system to protect memory and arbitrate between various I/O users.

applications have access to all I/O resources and all system memories; the operating system arbitrates access to these resources. Figure 5.25(b) illustrates these features.

Which programming model to use, SMP or AMP, depends on several factors such as the requirements of the application in terms of deadlines and throughput, and the application's characteristics, which may offer opportunity for parallel execution and may or may not have global and static variables and pointers. We must consider the structure of the existing code; is it modular or monolithic? Will two cores (or any two cores) be enough for future scalability? Then we must look at which hardware to select in terms of the types of cores, the memory architecture and interconnects and the available operating systems.

An example shown in Figure 5.26 illustrates how developers can apply these techniques to a quad-core system. In this example, each worker thread determines which portion of the array it should update, ensuring no overlap with other worker threads. All worker threads can then proceed in parallel, taking advantage of the SMP ability to schedule any thread on any available CPU core. Now that we have converted the software to a multithreaded approach, it can scale along with the number of CPUs. The example in Figure 5.26 uses a four-core CPU system, but it is easy to adjust the number of worker threads to accommodate more or fewer CPU cores.

A few tools that would come in handy for multicore software development are: (1) communication topology generators and configurators; (2) programming model tools; (3) tools for application partitioning, a potentially very complex task; (4) debuggers that can "focus" in and out to view information both at the detail and function levels, and that can stop execution on selective cores, without "breaking" the application; (5) simulators; (6) optimization tools.

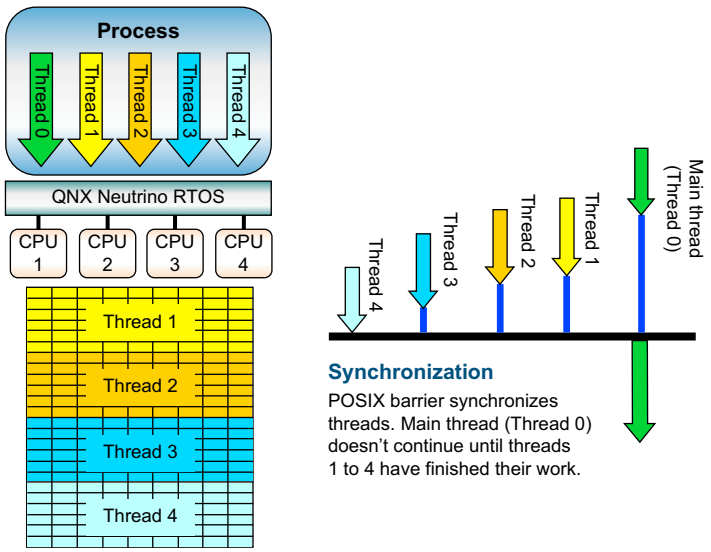


FIGURE 5.26

Multi-threaded approach in which each worker thread updates a portion of the array. Barrier synchronization ensures that all worker threads have finished updating the array before the main thread proceeds.

(Courtesy of QNX Software Systems.)

(1) Multicore operating systems

The operating system chosen for a multicore design depends on how the operating system supports the various multiprocessing modes that a multicore chip may offer. Based on the discussion earlier, these modes come in three basic paradigms: (1) asymmetric multiprocessing, in which a separate operating system, or a separate instantiation of the same operating system, runs on each processor core; (2) symmetric multiprocessing, in which a single instantiation of an operating system manages all processor cores simultaneously, and applications can float to any core; (3) bound multiprocessing, in which a single instantiation of an operating system manages all processor cores. Each mode has its own benefits and trade-offs; each is best suited to solving specific problems, and each makes unique demands on the operating system.

In SMP the programmer has no a priori knowledge about which processor a thread will execute on. The operating system must ensure inter-thread communication regardless of whether interacting threads are allocated to the same or different cores. The SMP operating system schedules tasks or threads on any available core. Since the operating system has all cores at its disposal, tasks or threads can truly execute in parallel. This approach provides the ability to speed up computation-intensive operations by using more than one core simultaneously. With SMP and the appropriate multithreaded programming techniques, adding more processing cores makes computation-intensive operations run faster. Also, developers can employ semaphores, mutexes, barriers, and other thread-level primitives to synchronize applications across cores. These primitives offer synchronization with much lower overhead than the application-level protocols required by AMP.

In AMP, on the other hand, each processor has its own independently executing instance of the operating system. The programmer determines on which core or cores a given thread will execute, and defines appropriate mechanisms for inter-thread communications. Threads are statically allocated to particular cores at design time. Asymmetric multicore architectures typically employ several inter-core communication mechanisms, such as inter-core buses, inter-core shared registers and shared memory. The inter-process communication channels will be structured in an implementation-agnostic manner, allowing the operating system to choose the most appropriate method of communication between two processes at runtime.

(a) Inter-core communication

Cores interact through inter-core communication, which is implemented using a combination of shared memory and interrupts. An AMP system requires that threads executing different cores can communicate with each other. This is enabled by special hardware inter-core communication support. These capabilities should be incorporated in such a way that their use in application development is very natural if this communication is to be effective. The goal is to provide an inter-core communication paradigm tailored to the multicore programming environment that is as simple and convenient to use as that of a single-core programming environment.

In a multicore environment a queue must be in shared memory. Once there, any thread on any core can write to or read from it, but, as was indicated for semaphores, it is not practical for a lightweight AMP operating system to allow a thread to be "pending" on a queue when other threads on other cores are "posting" to the queue. Instead, a queue is shared across cores must be polled to see if it has available data. The AMP operating system controls access to the shared queue by use of a spinlock.

In addition, it is important to ensure that queue initialization and use is appropriately synchronized across cores. A shared queue that is initialized by a single core may be used by many cores, but may

not be used safely until initialization is complete. This synchronization can be accomplished using either a spinlock or a barrier.

Inter-core messages provide another mechanism for a more active form of communication between cores. An inter-core message relies on the concept of a "virtual interrupt", a software-generated interrupt that is generated on core A but which acts on core B. The AMP operating system provides a mechanism for installing a message on a receiving core; in essence an ISR (interrupt service routine) for a particular virtual interrupt. This mechanism also activates a call-back function that can retrieve the message. Each virtual interrupt is associated with a data structure in shared memory.

To send a message to core B, core A initializes the data structure and "posts" the message. Core B receives an interrupt-generated call-back and is able to read the message. It is important to remember that the message must be read to avoid deadlock. To ensure that messages cannot be lost, they are locked by the sending core, ensuring that the message cannot be overwritten prior to being read. Upon reading the message, the operating system on the receiving core unlocks the message, enabling future messages to be sent.

Both "message" and "queue" mechanisms together provide the basic building blocks that enable more complex communication protocols to be constructed. A message and a queue may be used together to signal when a queue contains data. A back-to-back pair of messages (from core A to core B and then from core B to core A, for example) may be used to construct a bidirectional information flow, command/acknowledge structures, and data/answer protocols.

(b) Shared memory

Shared memory is a region of memory that is accessible by all of the cores in the system. It needs to provides a common reservoir for data that be shared across cores, including the storage for operating system constructs such as shared queues, messages, semaphores, etc. An application developer should take care only to use shared memory when threads sharing data are on different cores. Because shared memory is usually multi-ported, it is typically limited in size. It can also introduce execution stalls when there is contention for access.

(c) Interrupt control

Differently from single-core CPUs, interrupts may come from peripherals or may be generated internally by one of the other cores. Each interrupt should be forwarded to each core, and each core can respond appropriately if the interrupt is enabled for that core. When an interrupt is enabled on more than one core, the developer must take special care to implement an appropriate policy for clearing it. For instance, the interrupt could be cleared as soon as the first core responds. Alternatively, it could be cleared only after all responding cores have completed servicing the shared interrupt.

Two algorithms for multicore interrupt control have been developed: (1) software configurable interrupt distribution; (2) software triggered interrupt controls. The first algorithm uses the "enable" mechanism provided by both multicore hardware and software to obtain a load balance between cores by means of managing the IRQ (interrupt request queue) and FIQ (fast interrupt queue) for each core. The second, software triggered interrupt control, uses a "trigger" to write into a register in shared memory when an interrupt occurs. Then it sends interrupt information with implementing inter-core communication (door-bell algorithm) or various broadcast modes to all cores. Figure 5.27 gives a diagrammatic explanation for these two algorithms.

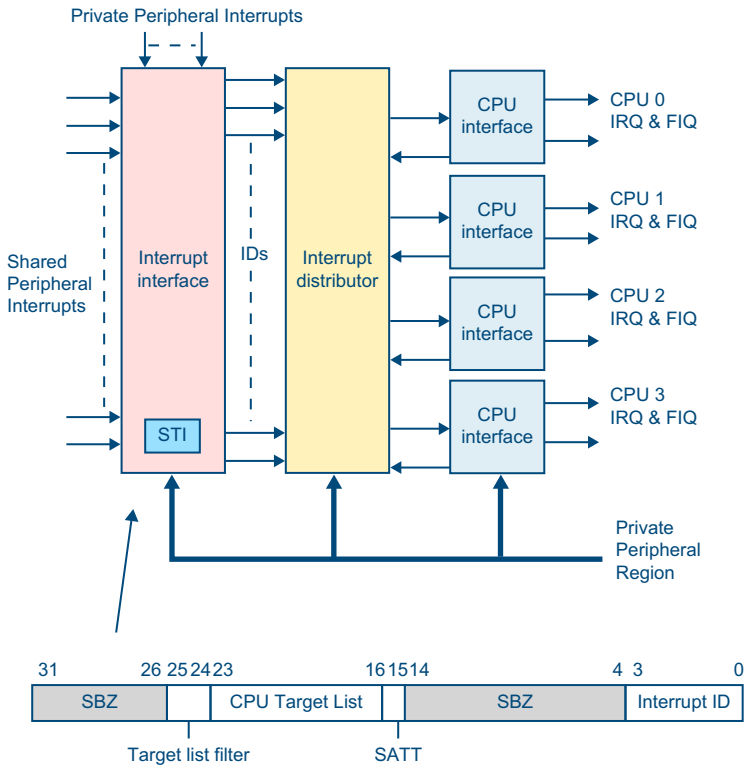


FIGURE 5.27

The dataflow of generic interrupt control for multicore chips. The left side is the interrupt interface for using software triggered interrupts; the middle box is the interrupt distributor for using software configurable interrupt distribution.

(d) Inter-core synchronization

Access to shared resources, such as shared memory and I/O peripherals, is controlled through spinlocks and semaphores. In both cases, a flag is used to control access to a shared resource of some kind. Although spinlocks and semaphores are similar in effect, they are significantly different in operation. The primary difference between a semaphore and a spinlock is that only semaphores support a pending operation. In a pending operation, a thread surrenders control; in this case until the semaphore is cleared, and the operating system transfers the control to the highest-priority thread that is ready to run. In contrast, the thread does not surrender control while the spinlock is active. Instead, the spinlock "spins/trigger" in a loop, continually checking the lock to determine when it becomes free. Hence, the operating system cannot switch to a new thread while the spinlock is active.

(e) Scheduling I/O resources

To enable this single interface to be shared efficiently, the I/O driver must support capabilities for routing data appropriately. For example, the driver should support a list of transmit data frames in

shared memory. To transmit a data frame, a thread on a particular core requests an empty data frame from the operating system. This frame is filled by the thread and passed to the I/O port driver for transmission. Once the data are transmitted, they are returned to the empty frame pool, ready for retransmission.

Similarly, for reception of data, each core will register a MAC address for its own "virtual" I/O port. In this way the device contains a MAC address for each core, allowing data to be routed effectively. Using the MAC address for identification, the received frame is routed to a receive buffer list, and an interrupt is sent to the receiving core to indicate arrival of data. Thus, each core can be programmed as though it has its own I/O interface. The software executing on a given core does not need to know that it is sharing the interface. Thus, software development is as straightforward as it is for a single-core device.

(2) Multicore programming compilers

A natural use of a multicore CPUs is to explore the use of thread level parallelism to speed up an application, requiring it to be multithreaded. Most multithreaded programming uses a conventional imperative programming language plus the parallel-thread library. For example, by merging HPC (high-performance computing) FORTRAN with a "pthread" library developed by SUN Microsystems, we have a parallel programming tool for dual-core or quad-core CPUs architectures. Unfortunately, it is notorious for its high complexity, low productivity, and poor portability. The HPC community has long sought a high-level parallel programming language which offers high performance as well as high productivity when applied to multicore CPUs systems.

What developers ideally want to be able to do is to take a single-core program such as a C, C++ or FORTRAN program, pass it through an auto-parallelizing compiler and automatically get out a multicore program. The result would be as portable, reliable and easy to debug as the original. The advantage of this idea is that the compiler can split up the program among as many cores as are suitable for the project. It would therefore be possible to try out different combinations of cores and clock speeds to get the best performance and power consumption ratios.

However, single-core software contains large numbers of dependencies. These are situations where one part of the software must be executed after another part of the software, hence two parts cannot be executed at the same time. It might be possible for the programmer to remove those dependencies, either because they are false dependencies, or because the algorithm used has dependencies in it (so the programmer could use a different algorithm if they were writing the program for a multicore processor chip). The compiler cannot, therefore, automatically change the order of execution of the program such that elements of it are executed at the same time on different processors. Specific programming compilers multicore CPUs to implement threading parallelism are required.

For this purpose, corporations and institutions around the world have made a lot of effort to develop multicore programming compilers. (they are also called multicore parallel-programming systems. Such a system can include compiler, linker and runtime; but its kernel is still the compiler), which basically fall into these three categories:

(a) multicore programming compilers developed by parallel extension of existing programming compilers for multiprocessor and computer cluster environments (note: multiprocessor and computer clusters are not multicore CPUs);

(b) multicore programming compilers requiring a multicore programming API (application program interface) such as OpenMP and Multicore Framework, etc.;
(c) multicore programming compilers requiring the Hyper-threading technology, as developed by Intel Corporation.

(3) Multicore communication API

The performance of an application running on multicore CPUs depends on efficient data movement between the cores to maximize parallel operation (comparable to cache efficiency on a single-core processor). The communication that deals with data movements between cores can be done with a consistent API, such as the MCAPI from the Multicore Association, even if the underlying hardware and software changes. Such changes may include the number and types of cores, memory architectures, different physical (and logical) interconnects as well as operating systems.

Furthermore, most operating systems in multicore CPUs, except those that are SMP-enabled, control one processor. A multicore system may have multiple instances of one, or even different types of operating systems. This requires both efficient on-chip communication, and dynamic management above the operating system level. Commonly used communication stacks such as TCP/IP, UDP/IP are not well suited for communication in a closely distributed computing environment such as multicore.

Multicore communication is best handled by a lightweight inter-processor communication framework (IPCF) designed to address requirements which are specific to a closely distributed multiprocessing environment and with a consistent programming API (note: there are two kinds of multicore API: communication API and programming API) such as OpenMP and Multicore Framework. The programming API abstracts the underlying cores, memory architectures, interconnects and operating systems. Because a multicore environment is static from a hardware perspective (the number of cores is fixed) and interconnects can be considered reliable, a multicore IPCF does not require some of the dynamic functionality needed on the Internet. This static nature allows efficient lightweight IPCF implementations, such as PolyCore Software's Poly-Messenger (Figure 5.28).

There is a need for more multicore-specific tools for tasks such as topology generation, debugging, partitioning, optimization and others. The first-generation multicore debuggers are available from a few venders of different operating system and development tools vendors. It is important to be able to reconfigure the communication topology easily, both for optimization purposes and to reuse application code on multiple and future hardware platforms. Separating the topology definition from the application makes reconfiguration easier, and an application that has been partitioned can potentially be remapped to different cores without modification.

A structured approach makes separating the topology configuration and application feasible, and a tool that generates the specified topology makes it easier and less error-prone. The topology can be defined in terms of structure, such as the communication endpoints; the nodes, which are equivalent to a core; the channels and priorities as well as the amount of resources that are assigned to the communication "components". By using a higher-level language, such as XML, to define topology, it can easily be reconfigured, reused and also be integrated in higher-level tools.

An example of the dataflow from topology definition to application integration is outlined in Figure 5.29. In this figure, the topology file is processed by the topology generator, which outputs the communication software topology in C language. The programs written in C are compiled and linked with

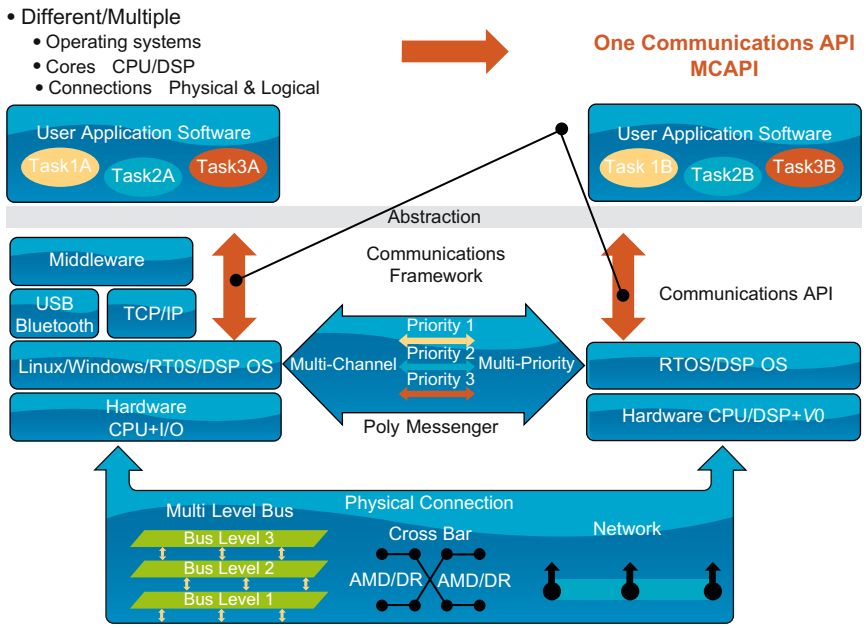


FIGURE 5.28

For abstracting the communication layers between cores, the underlying hardware communication mechanisms with a common API is used.

(Courtesy of the PolyCore Software.)

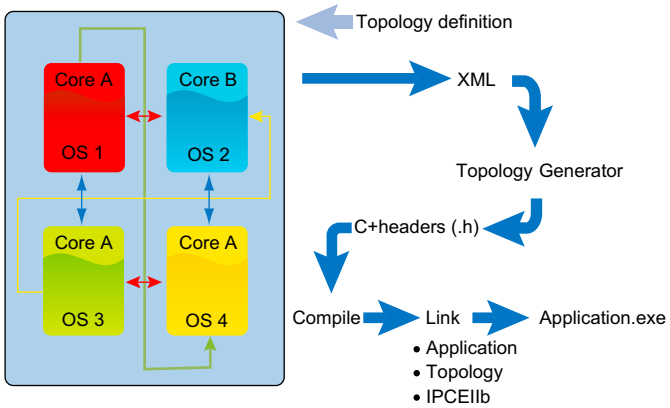


FIGURE 5.29

A topology generator enables the application and communication topology to be separated so that the communications can be reconfigured without requiring modifications to the application.

the application and the IPCF runtime library, resulting in a communication-enabled application. Topology changes are easily handled through modifications in the XML topology file. The topology is scalable (up and down) from one to even thousands of cores by using a topology generation tool that allows topology to be reconfigured outside your application, using an XML-based topology definition.

(4) Hyper-Threading Technology

Hyper-Threading Technology was developed by Intel Corporation to bring the simultaneous multi-threading approach to the Intel architecture. With this technology, two threads can execute on the same single processor core simultaneously in parallel, rather than context switching between the threads. As a result, a single physical processor appears as two logical processors; the physical execution resources are shared and the architecture state is duplicated for the two logical processors. From a software or architecture perspective, this means that operating systems and user programs can schedule processes or threads to logical processors as they would on multiple physical processors. From a micro-architecture perspective, this means that instructions from both logical processors will persist and execute simultaneously on shared execution resources.

Hyper-Threading Technology has been available on Intel processors such as Xeon processors and some Pentium 4 processors for some while. This technology adds circuitry and functionality into a traditional processor, each of which is then referred to as a logical processor. The added circuitry enables the processor to maintain two separate architectural states, and separate advanced programmable interrupt controllers (APICs), which provides multiprocessor interrupt management and incorporates both static and dynamic symmetric interrupt distribution across all processors. The shared resources include items such as cache, registers, and execution units, which can hence execute two separate programs or two threads simultaneously.

It is important to recognize that the requirements for enabling Hyper-Threading Technology are a system having a processor with Hyper-Threading Technology, an operating system that supports Hyper-Threading Technology and BIOS support to enable/disable Hyper-Threading Technology.

Note that it is also possible to have a dual-core processor system that contains two Hyper-Threading Technology enabled processors which would provide the ability to run up to four programs or threads simultaneously. This capability is available on Intel Xeon processors, and these systems are currently available from Intel Xeon processor-based data-processing systems.

Problems

1. Do some research to work out a chronology listing all generations of microprocessor evolutions (including the microprocessors' names and type, technical indices and specifications) from 2000.
2. Please write down your understanding of these three features of microprocessors: parallelism, prediction, and speculations.
3. Explain the functions and working mechanisms for both code caches and data caches with all the Intel Pentium processors.
4. What is MMX, and how does it work? Why is MMX used in microprocessors?
5. Plot the function blocks diagram for a typical Intel or AMD microprocessor.
6. What are interrupt vectors, interrupt vector tables, and interrupt service routines?
7. What are the differences between the isolated I/O technique and the memory mapped I/O technique used in microprocessor technology?

8. Please explain why both computer networks and computer clusters can be said to be "multiprocessors" systems, but cannot be said to be "multicore processor" systems.
9. This textbook gives two working strategies, snooping protocol and directory based protocol, to deal with "cache coherence". Please find other strategies for "cache coherence" and make a review of them.
10. Can you do some research to discover some places in real life where using parallel control to replace sequential control and using multicore to replace single core processors would enhance control performance factors such as efficiency, frequency, and safety.
11. Please give a definition for "system throughout" in industrial control technology, and explain how it is important in industrial control systems.
12. Based on Figures 5.15, 5.16, 5.17, and 5.18, explain the working principle for the Cell architecture of multicore processors.
13. Make a comparison between these three cache strategies for multicore processor systems: coherent cache, cooperative cache, and synergistic cache.
14. In addition to the shared bus fabric technology explained in this textbook, there are other bus architectures used in multicore chips. Please name one or two of them.
15. Please explain the advantages and disadvantages of SoC and NoC technologies for multicore hardware implementation.
16. What are the differences between "communication API" and "programming API" in multicore software implementation?

Further Reading

Zilog (http://www.zilog.com). 2005. http://www.zilog.com/docs/z180/ps0140.pdf. Accessed: June 2008.

InfoMit (http://www.informit.com). http://www.informit.com/articles/article.asp?p=482324&seqNum=3&rl=1. Accessed: June 2008.

Intel (http://www.intel.com). 2005a. http://www.intel.com/pressroom/kits/quickreffam.htm. Accessed: June 2008.

Intel (http://www.intel.com). 2005b. 8259A programmable interrupt controller: http://bochs.sourceforge.net/techspec/intel 8259a pic.pdf.gz. Accessed: June 2008.

Intel (http://www.intel.com). 2005c. 82C54 programmable interval timer. http://bochs.sourceforge.net/techspec/intel 82c54 timer.pdf.gz. Accessed: June 2008.

Intel (http://www.intel.com). 2005d. 8237A programmable DMA controller. http://bochs.sourceforge.net/techspec/intel 8237a dma.pdf.gz. Accessed: June 2008.

Best Microprocessor (http://www.best microcontroller projects.com). http://www.best microcontroller projects.com/hardware interrupt.html. Accessed date: June 2008.

Delorie (http://www.delorie.com). http://www.delorie.com/djgpp/doc/ug/interrupts/inthandlers1.html. Accessed: June 2008.

Styles, Lain. 2007. Lecture 11: I/O & Peripherals. http://www.cs.bham.ac.uk/internal/courses/sys arch/current/lecture11.pdf. Accessed: December 2007.

Datorarkitektur, I. 2007. Input/output device. http://www.ida.liu.se/~TDTS57/info/lectures/lect4.frm.pdf. Accessed: December 2007.

Decision Cards (http://www.decisioncards.com). http://www.decisioncards.com/io/index.php?style=isol. Accessed: December 2008.

Real Time Devices (http://www.rtdfinland.fi). DM5854HR/DM6854HR Isolated digital I/O Module User's Manual. Available from this company's website. Accessed: December 2008.

Intel (www.intel.com). Multi core processor architecture explained. http://software.intel.com/en us/articles/multi core processor architecture explained/. Accessed: January 2009.

Rogue Wave Software (www.roguewave.com). Homogenous vs. heterogeneous multi core for hardware strategies. http://www.roguewave.com/blog/rogue wave dropping support for apache c standard library. Accessed: January 2009.

CSA (www.csa.com). Multicore processor. http://www.csa.com/discoveryguides/multicore/review.php. Accessed: January 2009.

AMD (www.amd.com). AMD multicore processors technology: http://multicore.amd.com/. Accessed: January 2009.

SUN (www.sun.com). AMD multicore White Paper. http://www.sun.com/emrkt/innercircle/newsletter/0505multicore wp.pdf. Accessed: January 2009.

Rakesh Kumar, Dean M. Tullsen (University of California). Heterogeneous chip multiprocessors. IEEE Publication: 0018 9162/05. 2005.

QNX Software Systems (www.qnx.com). Kerry Johnson and Robert Craig: Software optimization techniques for multi core processors. 2007.

QNX Software Systems (www.qnx.com). Kerry Johnson and Robert Craig: Operating system support for multi core processors. 2005.

Basic Cell architecture. http://moss.csc.ncsu.edu/~mueller/cluster/ps3/doc/CellProgrammingTutorial/BasicsOfCellArchitecture.html. Accessed: January 2009.

Jacob Leverich et al. (Stanford University). Comparing memory systems for chip multiprocessors. ISCA Proceedings, SanDiego, California, USA. June 2007.

Rakesh Kumar et al. (University of California). Interconnections in multi core architectures: Understanding mechanism, overheads and scaling. IEEE Publication: 0 7695 2270 X/05. 2005.

IBM (www.ibm.com). IBM multicore research and design. http://domino.research.ibm.com/comm/researchprojects.nsf/pages/multicore.index.html. Accessed: January 2009.

Shameem Akhter, Jason Roberts (Intel Corporation). Multicore programming. Intel Press. (ISBN 0 9764832 4 6). 2006.

CHAPTER

Programmable-logic and application-specific integrated circuits (PLASIC)

6

Up to now, the advances in very large scale integration (VLSI) technology have brought chips containing billions of transistors into our laboratories, offices, and homes. In order to be competitive, companies have needed to develop new products and enhance existing ones by incorporating the latest commercial off-the-shelf VLSI chips; and increasingly by designing chips which are uniquely tailored for their own applications. These so-called application-specific integrated circuits (ASICs) are changing the way in which electronic systems are designed, manufactured, and marketed in all industries. Thus, it is important in this competitive environment to understand the nature, options, design styles, and costs of ASIC technology and of the related programmable-logic IC families.

ASIC is basically an integrated circuit that is designed specifically for an individual purpose or application. Strictly speaking, this also implies that an ASIC is built for one and only one customer. The opposite of an ASIC is a standard product or general-purposed IC, such as a logic gate or a general-purposed microcontroller chip, both of which can be used across a large range of electronic applications. ASICs are usually classified into one of three categories; full custom, semi-custom, and structured, as listed below.

(1) Full-custom ASICs are entirely tailor-made for a particular application from the out set. Since ultimate design and functionality are pre-specified by the user, they are manufactured with all the photolithographic layers of the device already fully defined, just like most off-the-shelf general-purposed ICs. A full-custom ASIC cannot be modified to suit different applications, and it is generally produced as a single, specific product for a particular application only.

(2) Semi-custom ASICs, on the other hand, can be partly customized to serve different functions within their general area of application. Unlike full-custom ASICs, they are designed to allow a certain degree of modification during the manufacturing process. A semi-custom ASIC is manufactured with the masks for the diffused layers already fully defined, so the transistors and other active components of the circuit are already fixed for that design. The customization of the final ASIC product to the intended application is done by varying the masks of the interconnection layers, for example, the metallization layers.

(3) Structured or platform ASICs, a relatively new ASIC classification, are those that have been designed and produced from a tightly defined set of (1) design methodologies, (2) intellectual properties, and (3) well-characterized silicon, aimed at shortening the design cycle and minimizing development costs. A platform ASIC is built from a group of "platform slices", with a platform slice being defined as a premanufactured device, system, or logic for that platform. Each slice used by the ASIC may be customized by varying its metal layers. The reuse of premanufactured and

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10006-3

precharacterized platform slices simply means that platform ASICs are not built from scratch, thereby minimizing design cycle time and costs.

Programmable-logic and application-specific ICs mean those ASICs that are built into programmed logic in advance. Programmable-logic and application-specific ICs can be found in a wide array of application areas including (1) an IC that encodes and decodes digital data using a proprietary encoding and decoding algorithm, (2) a medical IC designed to monitor a specific human biometric parameter, (3) an IC designed to serve a special function within a factory automation system, (4) an amplifier IC designed to meet certain specifications not available in standard amplifier products, (5) a proprietary system-on-a-chip (SoC) and network-on-chip (NoC), and (6) an IC that is custom-made for particular automated test equipment.

6.1 FABRICATION TECHNOLOGIES AND DESIGN ISSUES

This section identifies and discusses the underlying semiconductor technologies that are used to fabricate programmable-logic and ASIC devices. The design options, the design flow and synthesis technologies used for the PLASIC will be explained.

6.1.1 ASIC fabrication technologies

ASICs are normally fabricated using complementary metal oxide silicon (CMOS) technology. This subsection outlines the essential properties of CMOS devices, and shows how some of the basic circuit elements of an ASIC are implemented in CMOS. It first classifies the functional elements of an ASIC, and then gives an explanation of the ASIC fabrication process.

(1) Basic CMOS principles

The metal oxide silicon (MOS) transistor is a field effect device. Two types of MOS transistor are used on ASICs; n-types and p-types. These are identical in structure but have opposite polarity charge carriers, hence the term complementary metal oxide silicon. Modern devices use polysilicon gates and not metal, but the name has remained the same. The point to note is that the field effect transistors are enhancement devices. They require their gate voltage to be a threshold voltage, above their source in the case of the n-transistor, and below source for the p-type transistor. An n-type and a p-type together (Figure 6.1) form the inverter which is the most basic primitive to be found in an ASIC library. P-type and n-type transistors are used in pairs for all CMOS devices.

(1) CMOS implementation of primitive parts

The use of symbols to represent schematics as in Figure 6.1 is important. A symbol at this level represents a part whose schematic is a transistor configuration. It is called a primitive part because it is at the lowest level of the parts hierarchy, and cannot be decomposed into simpler parts. Higher-level parts have schematics made up of symbols from lower-level parts, including primitive. ASIC manufacturers supply a library of primitive parts represented as symbols. To some extent, designing with these symbols is rather like designing discrete logic from a data book of parts. This analogy should not be carried too far, because the principles of hierarchical ASIC design are not the same as those used when designing with discrete logic.

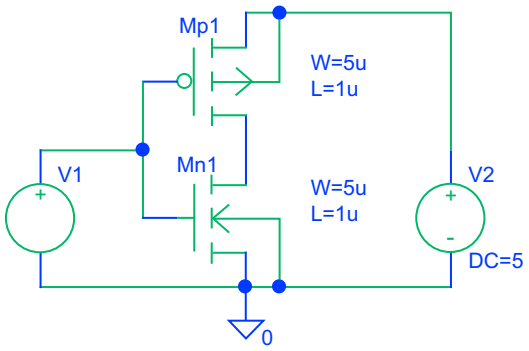


FIGURE 6.1

Schematic and symbols for CMOS inverter.

Another way of representing the characteristics of primitive parts is in terms of a switch-level model. The switch-level model of an inverter is another useful model up to quite high frequencies, typically above 20 MHz. It also helps in the understanding of such concepts as absolute and relative fan-out, drive strength and asymmetric drive capability.

A transistor can be configured into three gates: NAND, NOR, and ANDNOR Gates. In additional, comparators, decoders and inequality detectors form essential control elements in synchronous systems. They are related decoders perform a static, fixed comparison, whereas comparators perform a variable register-to-register comparison.

(2) Transmission gate and tristate buffer

Transmission gates represent a use of CMOS transistors that is not found in transistor-transistor-logic families. The transmission gate is an interesting primitive that is a true switch having two states; high impedance and low-impedance. However, great care must be taken to understand the nature and use of transmission gates, and how they can be transformed into triple state buffers.

(3) Edge-sensitive flip-flop

The edge-sensitive flip-flop is the basic storage element of a static, synchronous ASIC. Its essential property is that, on a rising clock edge, the logical state of input d is latched to output q, and that these output states persist until the next rising clock edge. A number of more complex latches, with feedback to retain data over successive clock cycles, and multiplexers to select the source of input data, are all derived from this basic cell. These form the basis of synchronous techniques for ASIC design.

(4) Classification of signals

All signals in a static synchronous ASIC can be classed as either clock, control or data.

1. The single clock signal is used to control all edge-sensitive latches, and for no other purpose. It is not gated with any other signal.

2. Control signals, such as enable and reset, are used to initialize circuit elements, hold them in their current state, and make choices between input signals or route signals to alternative outputs. They may all originate from a single enable generator, controlled by the state counter.
3. Data signals carry data, as individual bits or in buses, around the ASIC.

(5) Classification of base primitives

The set of primitive parts which are used to build up the hierarchical structures in a static synchronous ASIC can be classified as follows:

1. Boolean primitives which include: Inverter, AND, OR, exclusive OR (non-equivalence), NAND, NOR, exclusive NOR (equivalence) ANDNOR, ORNAND.
2. Switch primitives which include: transmission gate, multiplexer and triple state buffer.
3. Storage primitives.

Edge-sensitive flip-flop can include:

1. Control elements: decoder, comparator, inequality detector.
2. Data conditioning elements: adder, multiplier, barrel shifter, and encoder.

(2) The ASIC fabrication process

Having prepared the functional elements of an ASIC, we are now at the position to look at the ASIC fabrication process.

Those ASICs that are made for CMOS are manufactured by a process of repeatedly etching a chemical resist layer on to a silicon wafer to a required pattern, and then chemically diffusing, growing or etching layers. The resist layer is formed either by a photolithographic process, or by electron beam exposure. In general, photolithography is quicker, but has the disadvantage that mask contamination is replicated across the wafer. Other disadvantages are that the maximum chip size is limited by the final aperture in the lithographic equipment.

Device geometry (the minimum polysilicon track width possible with the process), is often quoted as a measure of the performance of a process. Small geometries, at present a 1.5 micron process, are faster than larger geometries such as 3 micron. Smaller geometries produce higher speeds due to lower gate capacitance, lower transition time across gates, and a higher packaging density which reduces track loading. Device geometry is only slightly dependent on whether the process is electric-beam or optical-beam in nature; the most variable component is the quality of the resist. As technology moves on, other limiting factors are likely to become more significant. The question of photolithography versus electric-beam is of more concern to accountants than to systems engineers, because it affects the unit price for various fabrication batch sizes.

During manufacture, process control monitors, or drop-ins are placed onto the wafer between customer designs. On a typical processor-control monitor there are raw transistors and via chains to test the parameters of the process for instances the threshold voltage. Some monitors for process control include RAM to test process yield, and a standard component such as a UART to test process performance at speed.

Approximately ten layers are required to create a working CMOS device. The top few layers are those of interest to the systems engineer; the two metal interconnect layers and the insulation layers (or via layers as they are known). The CMOS device can be regarded as a microscopic tray of transistors

with their legs in the air, connected by a double-layer printed circuit board. Between the individual chips there are scribe channels to allow the wafer to be cut into individual chips.

After fabrication and while still in wafers, the chips are tested for fabrication faults by running a set of test vectors developed as a part of the ASIC design process. These are applied by test rigs through probes which contact the pads on the chip. The chips which pass the tests are packaged, and their pads bonded to the package pins. The tests are repeated through the pins to identify packaging faults.

The percentage of the chips that pass the test is the yield of the wafer. The economics of ASIC production are heavily dependent on this yield. The physical manufacturing process is fine-tuned to maximize it, and a significant proportion of the ASIC design cycle is devoted to making the circuit easy to test, and developing test vectors which will identify as many faults as possible before the chip is packaged and placed into a system. The issue of testing ASICs is dealt with in subsection 6.1.3.

6.1.2 The ASIC design options

This subsection introduces the range of options and styles that are available for integrated circuit design. Because this chapter focuses on the programmable-logic design styles, this subsection places programmable logic in context. Full-custom design is a basic ASIC design, but is not covered below as its meaning needs little further explanation.

(1) Standard-cell design

A standard cell is a group of transistor and interconnect structures which provides a Boolean logic function (e.g., AND, OR, XOR, XNOR, inverters) or a storage function (flip-flop or latch). The simplest cells are direct representations of the elemental NAND, NOR, and XOR Boolean functions, although cells of much greater complexity are commonly used (such as a 2-bit full-adder, or mused D-input flip-flop). The cell's Boolean logic function is called its logical view; functional behavior is captured in the form of a truth table or Boolean algebra equation (for combinational logic), or a state transition table.

Figure 6.2 shows the layout of a standard cell. In reality, such a cell would be approximately 25 microns wide on an ASIC with λ (lambda) = 0.25 microns (a micron is 10^{-6} meter). Standard cells are stacked like bricks in a wall; the abutment box defines the "edges" of the brick. The difference between the bounding box and the abutment box is the area of overlap between the bricks. Power supplies (labeled VDD and GND in Figure 6.2) run horizontally inside a standard cell on a metal layer that lies above the transistor layers. Each shaded and labelled pattern represents a different layer. This standard cell has central connectors (the three squares, labelled A1, B1, and Z in Figure 6.2) that allow it to connect to others.

In the standard-cell design methodology, predefined logic and function blocks are made available to the designer in a standard-cell library. The standard-cell libraries of logical primitives are usually provided by the device manufacturer as part of their service. Usually their physical design will be predefined so they could be termed "hard macros". Typical libraries begin with gate level primitives such as AND, OR, NAND, NOR, XOR, inverters, flip-flops, registers, and the like. Libraries then generally include more complex functions such as adders, multiplexers, decoders, ALU, shifters, and memory (RAM, ROM, FIFOs, etc.). In some cases, the standard-cell library may include complex functions such as multipliers, dividers, microcontrollers, microprocessors, and microprocessor support functions (parallel port, serial port, DMA controller, event timers, real-time clock, etc.).

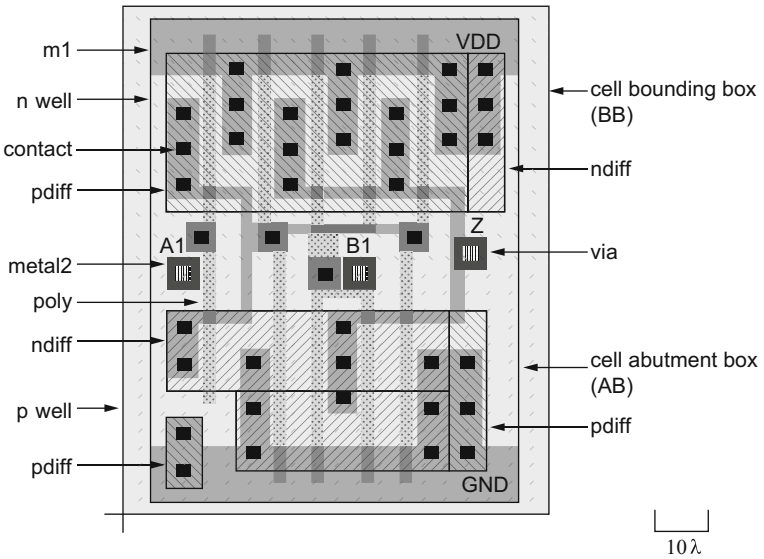


FIGURE 6.2

The layout of a standard-cell ASIC.

(Courtesy of Southeast University, China.)

Standard cell designs are created using schematic capture tools, or via synthesis from a hardware description language (HDL). Subsection 6.1.3 of this chapter will discuss the standard-cell design flow in more detail. Automated tools are then used to place the cells on a chip image and wire them together. Standard-cell designs operate at lower clock rates and are generally less area-efficient than a full custom design due to the fixed cell size constraints, and requirements for dedicated wiring channels. However, very high layout density is achieved within the cells themselves, resulting in densities which can approach that of full-custom designs, whilst needing substantially shorter design times.

(2) Gate-array design

Gate-array design is a manufacturing method in which the transistors and other active devices are predefined; and wafers containing such devices are held unconnected in stock prior to metallization. The physical design process then defines the interconnections of the final device. For most ASIC manufacturers, this consists of between two and five metal layers, each running parallel to the one below it. Non-recurring engineering costs are much lower, as photo-lithographic masks are required only for the metal layers, and production cycles are much shorter, as metallization is a comparatively quick process.

Pure, logic-only, gate-array design is rarely implemented by circuit designers today, having been replaced almost entirely by field-programmable devices, such as a field-programmable gate array (FPGA), which can be programmed by the user. Today gate arrays are evolving into structured ASICs that consist of a large ASIC IP core (ASIC IP means a semiconductor intellectual property) such as a CPU, DSP unit, peripherals, standard interfaces, integrated memories SRAM, and a block of reconfigurable uncommitted logic. This shift is largely because ASIC devices are capable of integrating such large blocks of system functionality, and because a "system on a chip" requires far more than just logic blocks.

In their frequent usage in the field, the terms "gate array" and "semi-custom" are synonymous. Process engineers more commonly use the term semi-custom, while gate array is more commonly used by logic (or gate-level) designers.

(3) Structured/platform design

Structured ASIC design (also referred to as platform ASIC design) is a relatively new term in the industry, which is why there is some variation in its definition. However, the basic premise of a structured/platform ASIC is that both manufacturing cycle time and design cycle time are reduced compared to cell-based ASIC, since the metal layers are predefined (thus reducing manufacturing time) and the silicon layer is pre-characterized (thus reducing design cycle time).

One other important aspect of structured/platform ASIC is that it allows IP that is common to certain applications or industry segments to be "built in", rather than "designed in". By building the IP directly into the architecture the designer can again save both time and money compared to designing IP into a cell-based ASIC.

(4) Field-programmable logic

A field-programmable-logic device is a chip whose final logic structure is directly configured by the end user. By eliminating the need to cycle through an IC production facility, both time to market and financial risk can be substantially reduced. The two major classes of field-programmable logic, programmable-logic devices (PLD) and FPGA, have emerged as cost-effective ASIC solutions because they provide low-cost prototypes with nearly instant "manufacturing". This class of device consists of an array of uncommitted logic elements whose interconnect structure and/or logic structure can be personalized on-site according to the user's specification.

6.1.3 The ASIC design flows

Nowadays, the design flow of ASIC has been highly automated. The automation tools provide reasonable performance and cost advantage over manual design processes. Broadly, ASIC design flow can be divided into the phases given below in this subsection, which are also illustrated in Figure 6.3.

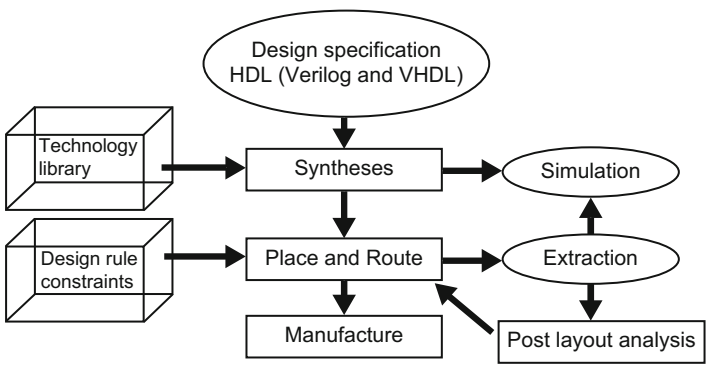


FIGURE 6.3

A generic ASIC design flow.

Designing a programmable-logic or application-specific IC involves many different tasks, some carried out by the ASIC designers, while others are taken care of by the ASIC vendor. Since a digital ASIC cannot be changed once it is manufactured, verification of its functions is crucial before the masks for production are made. Therefore, considerable effort is needed to simulate the design at different levels to verify that no bugs exist. The ASIC design is described using a hardware description language, HDL. There are a number of such languages available, such as Verilog, VHDL, System C or System Verilog. The language can describe the hardware at different levels of detail. The most common level used today is called register transfer level, RTL. This level describes the functions of the ASIC with logic relations between memory elements (registers).

Simulation is a key step for the programmed functions to verify their correctness, but they also need to be translated into hardware, i.e. logic gates. This is called synthesis and is done using software tools. The result of the synthesis must be verified, both from a functional and a timing perspective. This is done by gate-level simulations, or by comparing the logic produced by synthesis to the programmed functions in the HDL. The latter is called formal verification or equivalence check. A floorplan/layout tool places the logic gates on a model of the physical ASIC. This enables the calculation of signal delays between all logic elements.

Due to an increase in signal speed, miniaturization of features, smaller chip sizes, and lower power supply voltages, interconnect signal integrity problems have increased. Signal delay due to interconnect delay is more significant than gate delay. As a result, more powerful automation tools are required for layout parameter extraction, timing delay and crosstalk simulation, and power analysis. Static timing analysis (STA) is also used to constrain the layout tools and to verify that the design will work at the specified frequency. When the ASICs are manufactured, each chip needs to be tested in order to ensure that all delivered devices work properly.

The ASICs are tested using test vectors; either functional vectors or automatically generated vectors, or both. Functional vectors apply signals to the ASIC which are similar to the signals that the ASIC would receive in a real environment. Automatically generated vectors, on the other hand, only apply signals to verify that no defects exist within the ASIC, without relating to how the ASIC will be used in its target environment. This area of work is called design for test.

To gain better understanding of the ASIC design flow, design issues for the standard cell are taken as an example, and discussed in the following paragraphs. Usually, the initial design of a standard cell is developed at the transistor level, in the form a transistor netlist. This is a nodal description of transistors, of their connections to each other, and their terminals (ports) to the external environment. Designers use computer-aided design (CAD) programs to simulate the electronic behavior of the netlist, by declaring input stimulus (voltage or current waveforms) and then calculating the circuit's time domain (analogue) response. The simulations verify whether the netlist implements the requested function, and predict other pertinent parameters such as power consumption or signal propagation delay. Figure 6.4 is a physical design flow for standard-cell ASIC.

Standard-cell ICs are designed in the following conceptual steps (or flow), although they overlap significantly in practice. These steps, implemented with the level of skill common in the industry, almost always produce a final device that correctly implements the original design, unless flaws are later introduced by the physical fabrication process.

1. The design engineers start with a non-formal understanding of the required functions for a new ASIC, usually based on a customer requirements analysis.

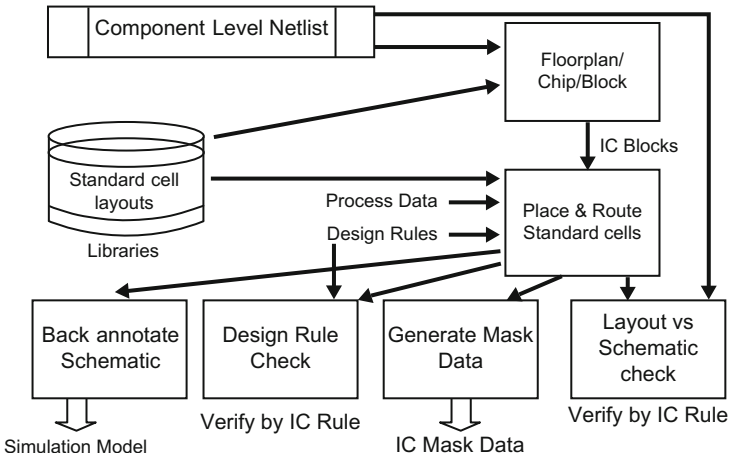


FIGURE 6.4

A physical design flow for a standard-cell ASIC.

2. The design engineers construct a specification of an ASIC to achieve these goals using an HDL. This is usually called the RTL (register transfer level) design.
3. Suitability for purpose is verified by functional verification. This may include such techniques as logic simulation, formal verification, emulation, or creating an equivalent pure software model.
4. Logic synthesis transforms the RTL design into a large collection of lower-level constructs called standard cells. These constructs are taken from a standard-cell library consisting of precharacterized collections of gates. The standard cells are typically specific to the intended manufacturer of the ASIC. The resulting collection of standard cells, plus the necessary electrical connections between them, is called a gate-level netlist.
5. The gate-level netlist is next processed by a placement tool which places the standard cells onto a region representing the final ASIC. It attempts to find a placement of the standard cells, subject to a variety of specified constraints.
6. The routing tool takes the physical placement of the standard cells and uses the netlist to create the electrical connections between them. The output is a file which can be used to create a set of photo-masks enabling a semiconductor fabrication facility to produce physical ICs.
7. Given the final layout, circuit extraction computes the parasitic resistances and capacitances. In the case of a digital circuit, this will then be further mapped into delay information, from which the circuit performance can be estimated, usually by static timing analysis. This and other final tests, such as design rule checking and power analysis (collectively called signoff), are intended to ensure that the device will function correctly over all extremes of the process, voltage and temperature. When this testing is complete the photo-mask information is released for chip fabrication.

These design steps (or flow) are also common to standard product design. The significant difference is that the latter uses the manufacturer's cell libraries that have been used in potentially hundreds of other design implementations, and therefore are of much lower risk than full custom design. Standard cells

produce a cost-effective, design density and they can also integrate IP cores and SRAM (static random access memory) effectively, unlike gate arrays.

6.2 FIELD-PROGRAMMABLE-LOGIC DEVICES

This section aims to provide a basic understanding of the designs, technologies and options available in the modern field-programmable-logic devices; field-programmable gate arrays (FPGAs), mask programmable gate arrays (MPGA), and programmable-logic devices (PLD), which include the simple PLD and complex PLD.

6.2.1 Field-programmable gate arrays (FPGA)

Field-programmable gate arrays (FPGA) are integrated circuits (ICs) that contain an array of logic cells surrounded by programmable I/O blocks. An FPGA can contain tens of thousands of programmable logic components called logic blocks or logic elements, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together", somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or be merely simple logic gates such as AND, XOR and OR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

Xilinx co-founders, Ross Freeman and Benard Vonderschmitt, invented the first commercially viable, field-programmable gate array, the XC2064, in 1985. This had programmable gates and programmable interconnects between gates, and marked the beginnings of a new technology and market. Xilinx continued unchallenged and grew quickly from 1985 to the mid 1990s when competitors started to erode their market-share. By 1993, Actel, as one of competitors to Xilinx, was serving about 18 percent of the market.

In the early 1990s, FPGAs were primarily used in telecommunications and networking. By the end of the decade, FPGAs had found their way into automotive and industrial applications. A recent trend has been to take the coarse-grained architectural approach a step further, by combining the logic blocks and interconnects of traditional FPGAs with embedded microprocessors and related peripherals, to form a complete "system on a programmable chip". An alternate approach to using hard-macro processors is to make use of the "soft" processor cores that are implemented within the FPGA logic.

Many modern FPGAs have the ability to be reprogrammed at runtime, which, as mentioned in the first paragraph of this subsection, leads to the idea of reconfigurable computing or reconfigurable systems. However dynamic reconfiguration at run-time is not supported, but instead the FPGA adapts itself to a specific program. Additionally, new, non-FPGA architectures are beginning to emerge. Software-configurable microprocessors adopt a hybrid approach by providing both an array of processor cores and also FPGA-like programmable cores on the same chip.

(1) Types and applications

From the application point of view, a field-programmable gate array (FPGA) is a semiconductor device that can be configured by the customer or designer after manufacturing, hence the name field-programmable. They are programmed by using a logic circuit diagram, or source code in a hardware

description language (HDL). They can be used to implement any logical function that an ASIC could perform, but have the ability to update the in functionality after shipping, which offers advantages for many applications.

Two types of FPGAs have emerged. Firstly, reprogrammable (or multiple programmable) FPGAs including SRAM-based and EEPROM-based versions, and secondly non-reprogrammable (or one-time programmable) FPGAs, which are either antifuse-based or EPROM-based. Table 6.1 gives a technical overview.

Traditionally, FPGAs have been reserved for specific vertical applications where the volume of production is small. For these low-volume applications, the premium that companies pay in hardware costs per unit for a programmable chip is more affordable than the development resources spent on creating an ASIC. At present, FPGAs are used in applications ranging from data processing and storage, to instrumentation, telecommunications, and digital signal processing. Other terms for FPGA include logic cell array and programmable application-specific integrated chip.

FPGA architecture offers massive parallelism. This allows for considerable computational throughput even at a low clock rates. The inflexibility allows for even higher performance, by trading-off precision and range in the number format for an increased number of parallel arithmetic units. This has driven a new type of processing, called reconfigurable computing, where time-intensive tasks are offloaded from software to FPGAs. The adoption of FPGAs in high-performance computing is currently limited by the complexity of FPGA design that is possible, in comparison with conventional software. The extremely long turn-around times of current design tools, where a 4 8-hour wait is necessary after even minor changes to the source code is also a limiting factor.

Field-programmable gate arrays are available with different numbers of system gates, shift registers, logic cells, and lookup tables. Logic blocks or logic cells do not include I/O blocks, but generally contain a lookup table to generate any function of inputs, a clocked latch (flip-flop) to provide registered outputs, and control logic circuits for configuration purposes.

Table 6.1 FPGA Types

Programming Technology	Configuration Technology	Technology Features
Reprogrammable FPGAs (multiple-time programmable)	SRAM-based FPGAs	An external device (SRAM: static-RAM, nonvolatile memory or μP) programs the device on power up. It allows fast reconfiguration. Configuration is volatile. Device can be reconfigured in circuit.
	EEPROM-based FPGAs	Configuration is similar to EEPROM (electrically erasable programmable read-only memory) devices. Configuration is nonvolatile. Device must be configured and reconfigured out of circuit (off-board)
Non-reprogrammable FPGAs (one-time programmable)	Antifuse-based FPGAs	Configuration is set by "burning" internal fuses to implement the desired functionality. Configuration is nonvolatile and cannot be changed.
	EPROM-based FPGAs	Configuration is similar to EPROM (erasable programmable read-only memory) devices. Configuration is nonvolatile. Device must be configured out of circuit (off-board).

Logic cells are also known as logic array blocks, logic elements, or configurable logic blocks. Lookup tables or truth tables are used to implement a single logic function, by storing the correct output logic state in a memory location that corresponds to each particular combination of input variables.

FPGAs are available with many logic families, transistor-transistor logic and related technologies. By contrast, emitter coupled logic uses transistors to steer current through gates that compute logical functions. Another logic family, CMOS, uses a combination of p-type and n-type metal-oxide-semiconductor field effect transistors to implement logic gates and other digital circuits. Logic families for FPGAs include crossbar switch technology, gallium arsenide, integrated injection logic, and silicon on sapphire. Gunning with transceiver logic and gunning with transceiver logic plus are also available.

FPGAs are available in a variety of IC package types and with different numbers of pins and flip-flops. Basic IC package types for field-programmable gate arrays include ball grid array, quad flat package, single in-line package, and dual in-line package. Many packaging variants are available.

As their size, capabilities, and speed increase, FPGAs are fulfilling functions, to the point where some are now marketed as full systems on chips (SoC).

(2) Architectures and designs

The hardware architecture, of FPGAs consists of an array of programmable logic blocks, routing matrix and global signals, input/output pads, multiplier, clock resources, memory cells and other advanced features. Figure 6.5 illustrates a generic example. Figure 6.6 gives an Actel FPGA function

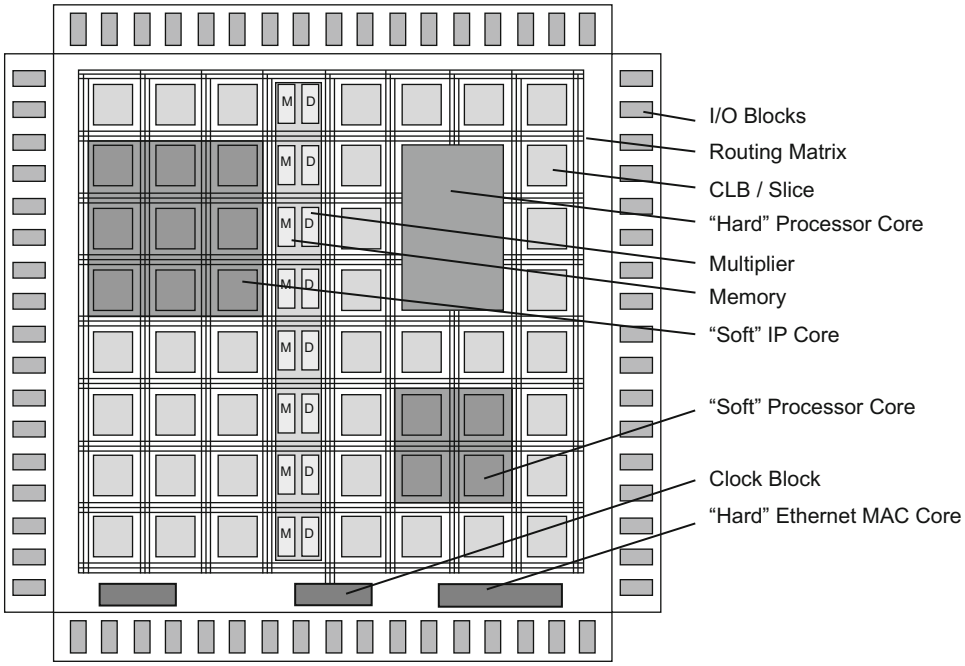


FIGURE 6.5

A generic FPGA hardware architecture.

as a basic example, showing several elemental components; logic modules/blocks, routing channels, I/O modules, channeled interconnects, and clock resources.

(1) Logic modules/blocks

In a FPGA chip, the logic module should provide the user with the correct mixture of performance, efficiency, and ease of design required to implement the application. The logic module must be therefore be optimized to ensure that the many conflicting goals of the designer are achievable.

1. Simple logic module. The first Actel logic module was the Simple logic module, used in the ACT-1 family. Shown in Figure 6.7, it is a multiplexer-based logic module. Logic functions are implemented by interconnecting signals from the routing tracks to the data inputs and select lines of the multiplexers. Inputs can also be tied to a logical 1 or 0 if required, since these signals are always available in the routing channel. A surprising number of useful logic functions can be implemented with this module, providing the designer with an excellent mixture of logic capabilities. Figure 6.8

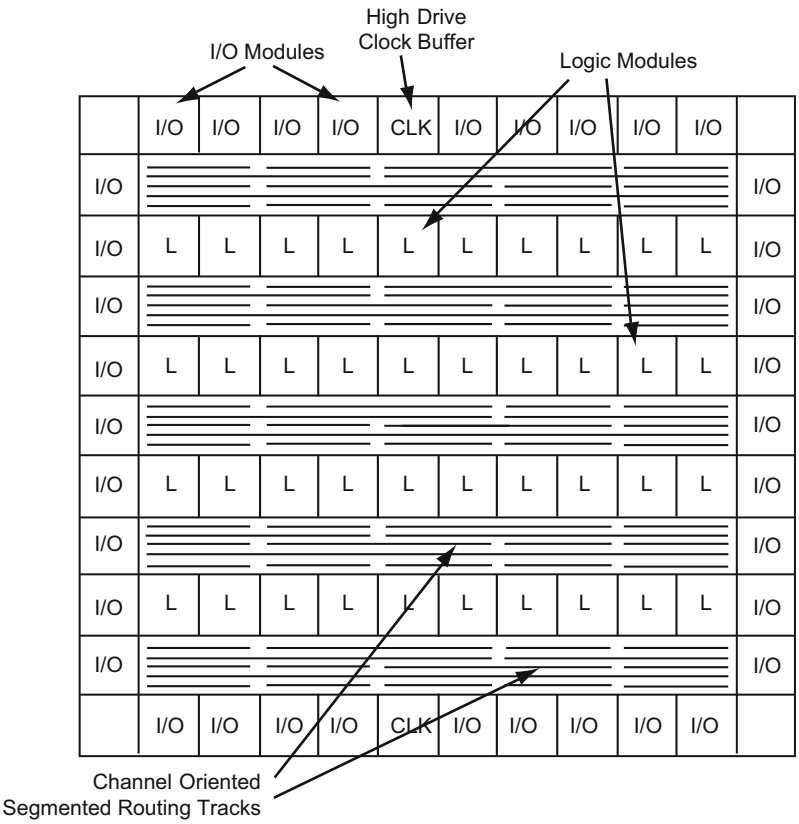


FIGURE 6.6

A basic Actel-FPGA function structure.

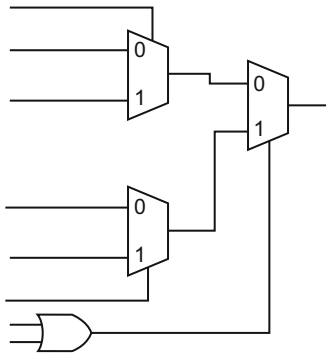


FIGURE 6.7

Simple logic module in FPGA design.

shows an example logic function implemented with the Actel simple logic module. Notice that latches can be implemented in a single logic module per bit and that registers require two logic modules per bit.

2. Combinatorial logic module. Some improvements were made to the simple logic module by replacing the simple logic module with two different logic modules, one for implementing combinatorial logic, (the combinatorial logic module) and one for implementing storage elements (the sequential logic module). The combinatorial logic module, shown in the diagram in Figure 6.9, is similar to the simple logic module, but an additional logic gate is placed on the first-level multiplexer. The added gate improves the implementation of some combinatorial functions. (Some five-input gates are now available.) Also, the first-level multiplexer lines in the simple logic module were combined in the combinatorial logic module. In the simple logic module, the separate multiplexer select lines were used to implement latches and registers efficiently. This was not required in the combinatorial logic module because of the addition of the sequential logic module. Figure 6.10 shows an example of a logic function implemented with the combinatorial logic module.

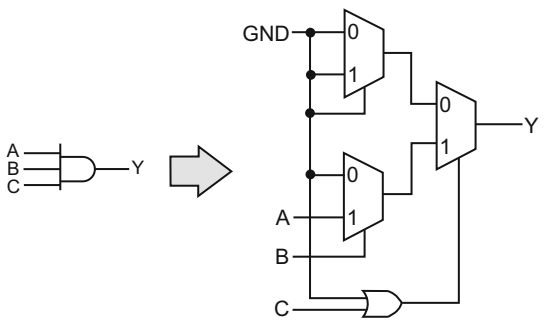


FIGURE 6.8

An example of a logic function implemented with the Actel ACT-1 simple logic module in FPGA design.

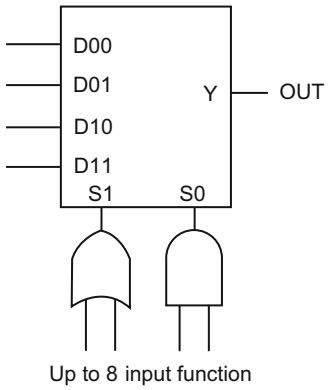


FIGURE 6.9

Combinatorial logic module in FPGA design.

3. Sequential logic module. The sequential logic module, shown in Figure 6.11, has a combinatorial logic front-end with a dedicated storage element on the output. The storage element can be either a register or a latch. (It can also be bypassed so the logic module can be used as a combinatorial logic module.) The clock input can be either active high or active low. One of the logic gates is missing on the combinatorial logic diagram, making it slightly different from the combinatorial logic module. The exclusion of this one logic gate allows the reset signal, which is shared with the combinatorial logic section, to be made available to the storage element without increasing the number of total module inputs required. If the storage element is bypassed, the reset signal is used to implement the required combinatorial module input. In the Integrator Series, sequential and combinatorial modules are interleaved, resulting in a 50 50 mix of logic modules. This has been found to be an optimal mix for a wide variety of designs, and results in excellent utilization.

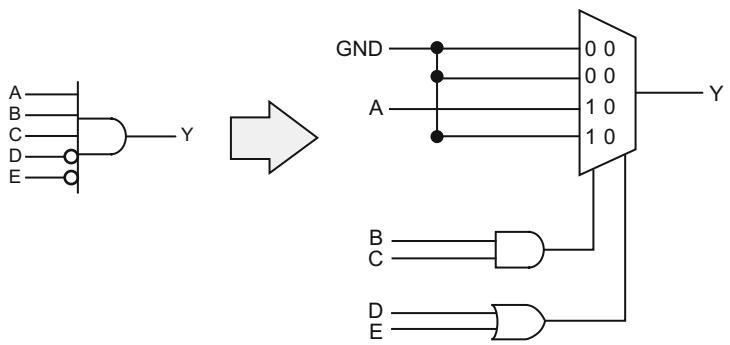


FIGURE 6.10

An example of a logic function implemented with the combinatorial logic module in FPGA design.

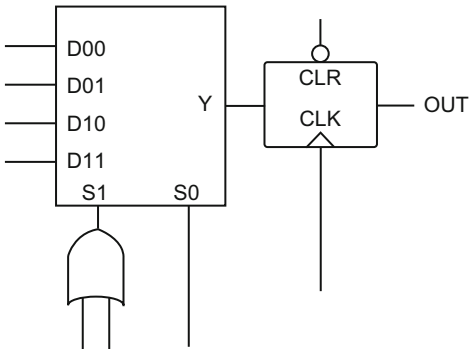


FIGURE 6.11

Sequential logic module in FPGA design.

The key to a high-performance, area-efficient architecture is offered by an advanced logic module, adaptive logic module, which consists of combinational logic, two registers, and two adders as shown in Figure 6.12. The combinational portion has eight inputs and includes a lookup table, which can be divided between two tables using patented lookup technologies. An entire adaptive logic module is needed to implement an arbitrary six-input function, but because it has eight inputs to the combinational logic block, one adaptive logic module can implement various combinations of two functions. Lookup tables are built to help the description of the adaptive logic module. This is typically built out of SRAM bits to hold the configuration memory (CRAM) lookup table-mask and a set of multiplexers

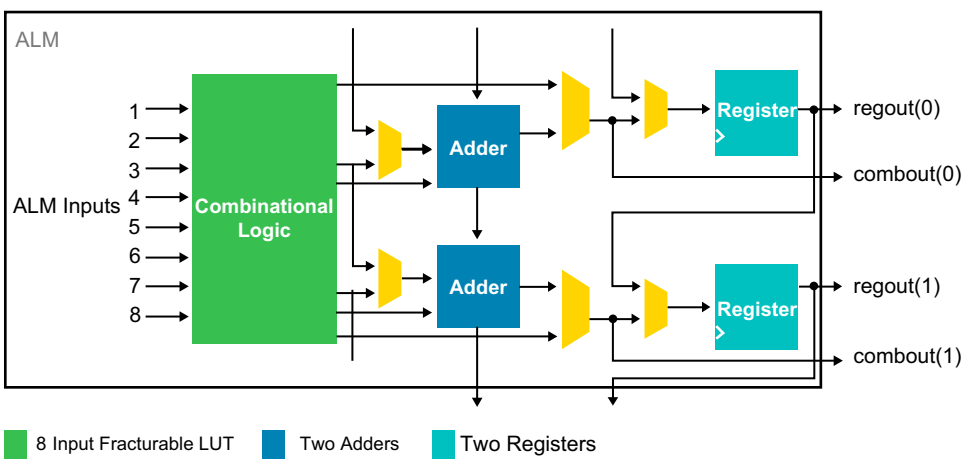


FIGURE 6.12

Adaptive logic module block diagram for FPGA design.

(ALM in the figure means adaptive logic module)

to select the bit of CRAM that is to drive the output. To implement a k-input look-up table (k-lookup table is a lookup table that can implement any combination of k inputs) $2k$ SRAM bits and a $2k$:1 multiplexer are needed. Figure 6.13 shows a 4-lookup table, which consists of 16 bits of SRAM and a 16:1 multiplexer implemented as a tree of 2:1 multiplexers. The 4-lookup table can implement any function of 4 inputs (A, B, C, D) by setting the appropriate value in the Look-up table-mask. To simplify the 4-Lookup table in Figure 6.13, it can also be built from two 3-lookup tables connected by a 2:1 multiplexer.

(2) Routing tracks

In addition to logic block architecture, another key FPGA feature is its routing architecture, which provides connectivity between different clusters of logic blocks, called logic array blocks. It is measured by the number of "hops" required to get from one logic array block to another. The fewer the number of hops, and more predictable the pattern, the better the performance and the easier it is for CAD (computer-aided design) tool optimization. Routing is organized as wires in a number of rows and columns.

There are two primary routing architectures in FPGAs; segmentation and buffer. Due to space considerations, this textbook does not discuss their details.

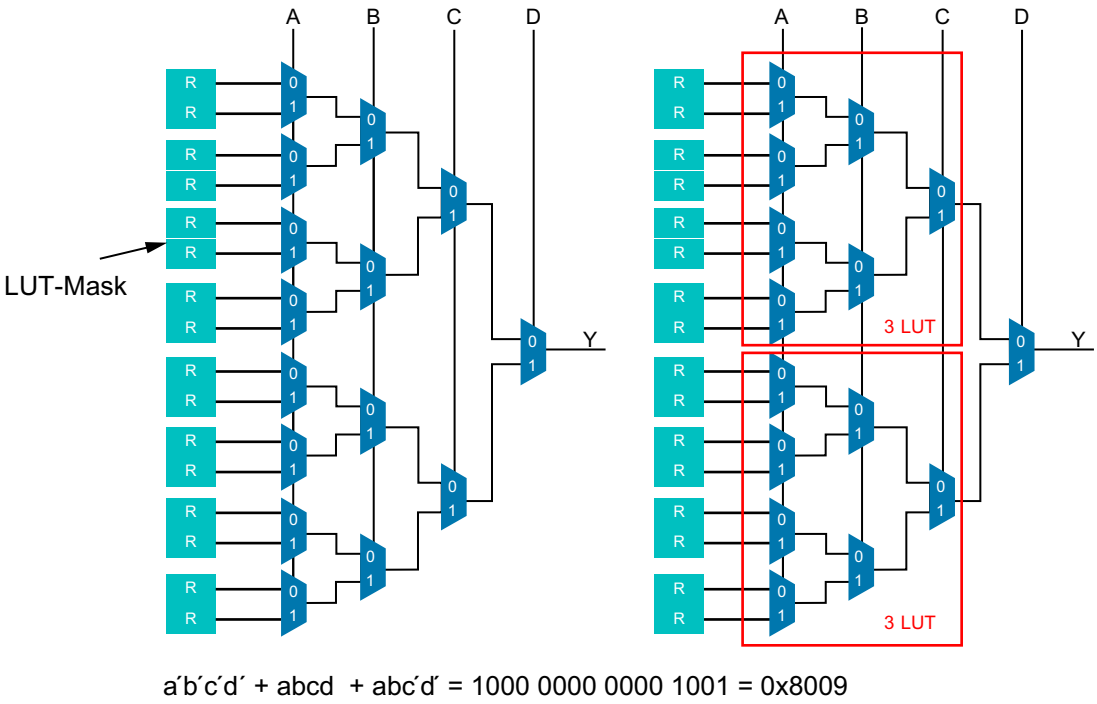


FIGURE 6.13

Building a lookup table to implement an adaptive logic module in FPGA design.

(LUT in the figure means lookup table)

The routing process is generally separated into two phases using the divide and conquer paradigm. They are a global routing that balances the densities of all routing channels, and a detailed routing that assigns specific wiring segments for each connection. These two phases avoid congestion, and optimize the performance of the circuit, making sure all nets are routed such that wire length and capacitance on the path are minimized. By running both algorithms, a complete routing solution can be created. There are a number of routing algorithms that solve the problem using mixed routing i.e. both global and detailed routing at the same time, based on the idea that a higher integration of the two phases can prevent inaccurate estimation. The drawback of this approach is that as circuit size grows, this mixed routing becomes more complex and less scalable.

(3) Channelled interconnects

With some FPGAs, the routing tracks are also defined as channelled interconnects. For example, all Actel FPGAs use a channeled interconnect architecture to make connections between internal logic modules and device I/O pins (Figure 6.6). This architecture is similar to that of a channeled gate array, in that horizontal tracks span the length of the array with a variety of predefined segmentation lengths. This makes a huge amount of routing resources available and ensures that signals usually have the length of available track that they require. In addition, tracks can be joined to construct longer tracks, when required, by programming an interconnect fuse. Logic module outputs span four channels (two above and two below as shown in Figure 6.6) and can be connected to any track. This means that most signals require only two fuses to connect any logic module output to any logic module input. There are enough routing resources available in Actel FPGA devices so that place and route is an automatic task. No hand routing is required. For more details on the interconnect resources available in Actel FPGA devices, refer to the device family data sheets.

(4) I/O modules

Each FPGA family, either from different manufacturers or from the different versions from the same a manufacturer, has a slightly different I/O module. However, all the I/O modules in FPGAs were developed from the simple I/O module, optimized as an advanced I/O module for a new balance of speed and cost (value). More details on each I/O module can be found in the associated device data sheets and application notes.

1. Simple I/O module. The simple I/O module is a simple I/O buffer with interconnections to the logic array. All input, output, and three-state control signals are available to the array. Outputs are TTL (transistor-transistor logic) and CMOS compatible and sink or source about 10 mA at TTL levels.

2. Latched I/O module. The latched I/O module is used in the Integrator Series and is slightly more complicated than the simple module. It contains input and output latches that can be used as such, or when combined with internal latches, become input or output registers. Outputs are TTL and CMOS compatible and sink or source about 10 mA at TTL levels.

3. Registered I/O module. The registered I/O module is optimized for speed and functionality in synchronous system designs. It contains complete registers at both the input and output paths. Data can be stored in the output register, or they can bypass the register if the control bit is tied low. Both the output and input register can be cleared or preset via the global I/O signal, and both are clocked via another global I/O signal. Notice that the output of the output register can be selected as an input to the

array. This allows state machines, for example, to be built right into the I/O module for fast clock-to-output requirements.

(5) Clock resources

Actel FPGA devices have a wide range of clocking flexibility. Every sequential element's clock input can be connected to regular interconnects within the channel, as well as to optimized clocking resources. Regular interconnects offer the most flexibility, allowing for thousands of potential separate clocks. Each Actel device also has dedicated clocking resources on-chip to improve clock performance and to simplify the design of sequential signals. Clocking resources can also be used, in most cases, as high-drive global signals such as reset, output enable, or select signals. Each FPGA family is slightly different in the way it implements clocking functions. For more details on each type of clocking resource, refer to the associated device data sheets and application notes.

1. Routed clocks. All Actel FPGA families have one or two special buffers that provide high-drive, low-skew signals and that can be used to drive any signal requiring these characteristics. These routed clocks are distributed to every routing channel and are available to every logic module. This allows a routed clock signal to be used by both sequential and combinatorial logic modules, offering maximum flexibility with slightly lower performance than dedicated clocks.

2. Dedicated array clock. The Actel's ACT-3 family has an additional clocking resource, consisting of a high-speed dedicated clock buffer that is optimized for driving sequential modules in the core array. This clock buffer can be driven from an external pin or from an internal signal. The dedicated array clock is optimized for driving sequential modules and cannot drive storage elements built from combinatorial modules.

3. Dedicated I/O clock. Another clocking resource consists of a high-speed dedicated clock buffer optimized for driving the sequential modules in the I/O modules. It is optimized for driving I/O modules and cannot drive storage elements in the array. If all storage elements need to be driven from a common clock, the array clock and I/O clock can be connected together externally.

4. Quad clocks. Some of the FPGA family has an additional clocking resource consisting of four special high-drive buffers called quadrant clocks. Each buffer provides a high-drive signal that spans about one-quarter of the device (a quadrant). These buffers can be used for fast local clocks (perhaps for pre-scaled shifters or counters), for wide-width selects, or for I/O enables. Note that since these are quadrant oriented, only a single quadrant clock can be used per quadrant. Quad clocks can be connected together internally to span up to one-half of the device. Additionally, the quad clocks can be sourced from internal signals as well as external pins. Thus they can be used as internally driven high fan-out nets.

(3) Programming and principles

Every FPGA relies on the underlying programming technology that is used to control the programmable switches to provide its programmability. There are a number available, and their differences have a significant effect on programmable-logic architecture. The approaches that have been used historically include EPROM, EEPROM, flash, SRAM (static-RAM), and antifuse programming technologies. Of these approaches, only the static memory, flash memory and antifuse approaches are widely used in modern FPGAs. The following will briefly review all modern FPGA programming technologies in order to provide a more comprehensive understanding of the FPGA working principle.

(a) Static memory programming technology

Static memory cells are the basis for SRAM programming technology, which is widely used, and can be found in the technical documents of FPGA superpowers such as Xilinx and Actel. In these devices, static memory cells are distributed throughout the FPGA to provide configurability. Most are used to set the select lines to multiplexers that steer interconnecting signals. An example of SRAM programming technology is shown in Figure 6.14, constructed from two cross-coupled inverters and using a standard CMOS process. The configuration cell drives the gates of other transistors on the chip by either turning pass transistors or transmission gates on to make a connection or off to break a connection.

The advantages of SRAM programming technology are the easy reuse of chips during prototyping. The technology is also useful for upgrades, where a customer can be sent a new configuration file to reprogram a chip, rather than a new chip. On the other hand, its disadvantage is the need to maintain the power supplied to the programmable ASIC (at a low level) for the volatile SRAM to retain its connection information. The total size of an SRAM configuration cell plus the transistor switch that the SRAM cell drives is also larger than the programming devices used in the antifuse technologies.

(b) Floating-gate programming technology

One alternative programming technology, floating-gate programming, can addresses some of the shortcomings of static memory programming, and hence is used for flash and EEPROM memories' programming. This floating-gate programming technology is achieved through a digital interface composed of a digital switch matrix and an analog/digital converter. Digital switches control the tunneling and injection voltages, and the digital decoders in order to provide individual access to the floating-gate transistors. An on-chip, specialized, analog/digital converter provides feedback to the programmer by outputting a digital signal with a pulse width that is proportional to the drain current of the floating-gate transistor currently being programmed. To avoid additional hardware on the prototyping station, the FPGA that is used to implement the digital part of the system in operational mode is also used to implement the programming algorithms in configuration mode.

(c) Antifuse programming technology

Antifuse programming is based on structures that exhibit very high resistance under normal circumstances, but can be programmably connected to create a low resistance link that is permanent. Different from SRAM and floating-gate programming technologies, the programmable

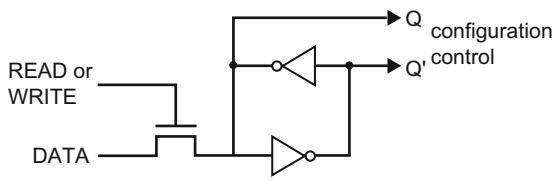


FIGURE 6.14

The Xilinx SRAM (static RAM) configuration cell.

element, an antifuse, is used directly for transmitting FPGA signals. Two approaches have been used to implement antifuse; dielectric and metal. Direct antifuses, identified at the center of Figure 6.15, are composed of an oxide-nitride-oxide dielectric positioned between N+ diffusion and polysilicon. The application of a high voltage causes the dielectric to break down, and form a conductive link with a resistance of typically between 100 and 600 ohms. The dielectric approach has been largely replaced by metal-to-metal-based antifuse. These antifuses are formed by silicon oxide between two metal layers. Again, a high voltage breaks down the antifuse and causes the fuse to conduct. The advantage of this method is that the on resistance can be between 20 and 100 ohms, and the fuse itself requires no silicon area. This metal-to-metal approach is used in recent FPGAs from Actel.

Antifuse technology is nonvolatile, so it is live at power-up and inherently very secure. Antifuse devices are mainly programmed using single-site or multi-site programmers. Types of programming for antifuse devices depend on the number and the type of devices to be programmed and are generally either device programmers or volume programming.

Device programmers are used to program a device before it is mounted on the system board, either before being soldered (usually done in production), or before putting it into a socket (used for prototyping). No programming hardware is required on the system board in this method, so no additional components or board space are required.

With the volume programming services, Actel can offer large volumes of parts which have been programmed, but programs that will not allow files to be sent off-site will not be able to use this approach. This includes Actel in-house programming, distributor programming centers, and independent programming centers.

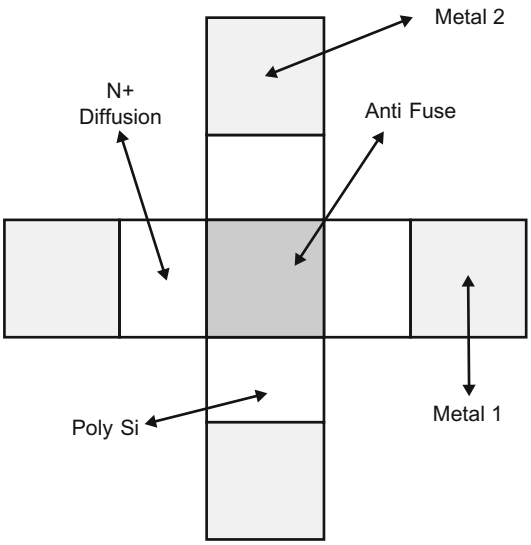


FIGURE 6.15

An antifuse programming interconnect.

Actel supplies two types of the programming software to serve for all the Actel programmers: single-site programmers and multiple-site programmers. Each version of the software enables users to select device, programming files, program, and verify the device.

6.2.2 Mask-programmable gate arrays (MPGA)

The mask-programmable gate array (MPGA) was developed to handle larger logic circuits. Although clearly not field-programmable devices, they did give considerable impetus to their indesign.

A common MPGA consists of several rows of transistors that can be interconnected in order to implement the desired logic circuits. User-specified connects are available, both within and between the rows. Customization is performed during chip fabrication by specifying the metal interconnect, which leads to setup costs long and high manufacturing times.

The most prevalent style of MPGA in current use is the sea-of-gates or sea-of-transistors architecture. The core of a sea-of-gates MPGA is a continuous array of transistors in fixed positions, surrounded by I/O circuits and bonding pads. Wafers containing the core design are pre-fabricated up to the final metallization steps. These master or base wafers are held in stock by the vendor until a customer design is received. Then, one or more custom masks are created to define the user's circuit with design specific metallization and contacts. Figure 6.16 shows the architecture of sea-of-gates MPGA.

6.2.3 Programmable-logic devices (PLD)

As already mentioned, logic devices can be classified into two broad categories; fixed and programmable. As the name suggests, the circuits in a fixed logic device are permanent, they perform one function or a set of functions, and once manufactured they cannot be changed. On the other hand, programmable-logic devices (PLDs) are standard, off-the-shelf parts that offer customers a wide range

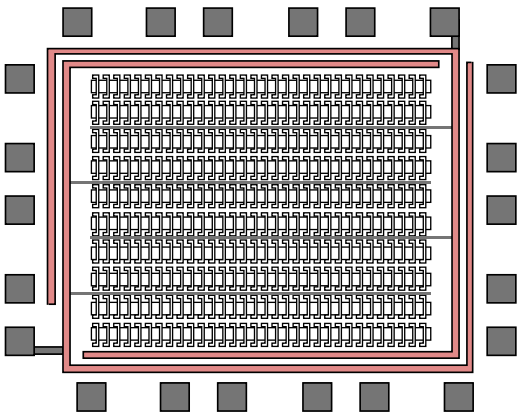


FIGURE 6.16

The sea-of-gates MPGA architecture.

of logic capacity, features, speed, and voltage characteristics; and hence these devices can be changed at any time (including at run-time) to perform any number of functions.

With fixed logic devices, the time required to go from design, to prototypes, to a final manufacturing run can between several months and more than a year, depending on the complexity of the device. If the device does not work properly, or if the requirements change, a new design then must be developed. With programmable-logic devices (PLDs), designers use inexpensive software tools to quickly develop, simulate, and test their designs. Then, a design can be quickly programmed into a device, and immediately tested in a live circuit. A key benefit of using PLDs is that customers can change the circuitry as often as they want during the design phase until it operates to their satisfaction.

Programmable-logic devices (PLDs) are designed with configurable logic and flip-flops linked together with programmable interconnects. They provide specific functions, including device-to-device interfacing, data communication, signal processing, data display, timing and control operations, and almost every other function a system must perform. Generally, PLDs are either simple programmable-logic devices (SPLDs), complex programmable-logic devices (CPLDs), or field-programmable-logic devices (FPGAs).

Programmable-logic devices are field-programmable gate arrays (FPGAs) or complex programmable-logic devices (CPLDs). The distinction between the two is often a little fuzzy, as manufacturers designing new, improved architectures. Together, CPLDs and FPGAs are often referred to as high-capacity programmable-logic devices.

Programming technologies for PLD devices are based on the different types of semiconductor memory. As new types of memories have been developed, the technology has been applied to the creation of new types of PLD devices major distinguishing feature between SPLDs, CPLDs, and FPGAs is the level of available logic. Today, SPLDs are devices that typically contain the equivalent of 600 or fewer gates, while CPLDs and FPGAs have thousands and more than several millions of gates available, respectively. FPGAs offer the highest amount of logic density, the most useful features, and the highest performance. They are used in a wide variety of applications, ranging from data processing and storage, to instrumentation, telecommunications, and digital signal processing. CPLDs, by contrast, offer much smaller amounts of logic up to about 10,000 gates, but they offer very predictable timing characteristics, and are therefore ideal for critical control applications. Some CPLDs require extremely low amounts of power and are very inexpensive, making them ideal for cost-sensitive, battery-operated, portable applications such as mobile phones and digital handheld assistants.

Because the commercially available SPLDs and CPLDs were and certainly will be change both their architectures and techniques very rapidly full, this subsection will not provide a list of all available types. Instead, this subsection will emphasize two key technologies used for programming and manufacturing; user-programming switch technologies and the ASIC packaging technologies, even though some features are common to FPGAs.

(1) User-programming switch technologies

For higher-density devices, where CMOS dominates the ASIC industry, different approaches to implementing programmable switches have been developed. For CPLDs the main switch technologies (in commercial products) are floating-gate transistors like those used in EPROM and EEPROM, and for FPGAs they are SRAM and antifuse. Each of these is briefly discussed below.

An EEPROM or EPROM transistor is used as a programmable switch for CPLDs (and also for many SPLDs) by placing the transistor between two wires to implement wired AND functions. This is illustrated in Figure 6.17, which shows EPROM transistors as they might be connected in an AND-plane of a CPLD. An input to the AND-plane can drive a product wire to logic level "0" through an EPROM transistor, if that input is part of the corresponding product term. For inputs that are not involved in a product term, the appropriate EPROM transistors are programmed to be permanently turned off. A diagram for an EEPROM based device would look similar.

Although there is no technical reason why EPROM or EEPROM could not be applied to FPGAs, current products are based on either SRAM or antifuse technologies, as discussed above.

An example of SRAM-controlled switches is illustrated in Figure 6.18, showing SRAM cells controlling the gate nodes of pass-transistor switches, and the select lines of multiplexers that drive logic block inputs. The figure gives an example of the connection of one logic block (represented by the AND-gate in the upper left corner) to another through two pass-transistor switches, and then to a multiplexer, all controlled by SRAM cells. Whether an FPGA uses pass-transistors, or multiplexers, or both depends on the particular product.

(2) ASIC packaging technologies

The ASIC package must provide electrical connections to the chip for both signal and power transfer. The package must also physically support the relatively small and fragile ASIC die, and must protect it from moisture, dust, gases, and other potential contaminants. Finally, the package must provide heat transfer from the die to the ambient environment or to the second-level package in order to prevent performance and reliability degradation.

In fact, ASIC packaging can affect system performance as much or more than the selection of ASIC design style or process technology. These influences upon system performance can be summarized as follows:

1. Degraded electrical performance (speed and power).
2. Increased size and weight.
3. Reduced testability.

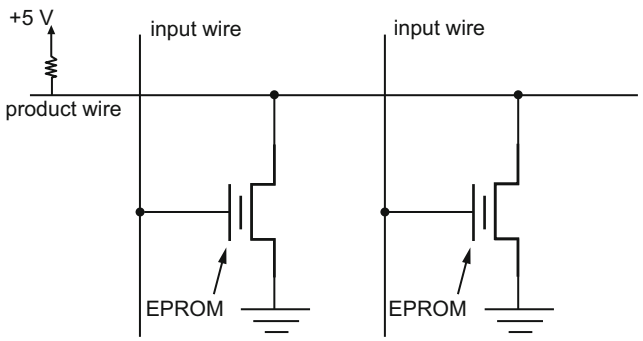


FIGURE 6.17

EPROM programmable switches.

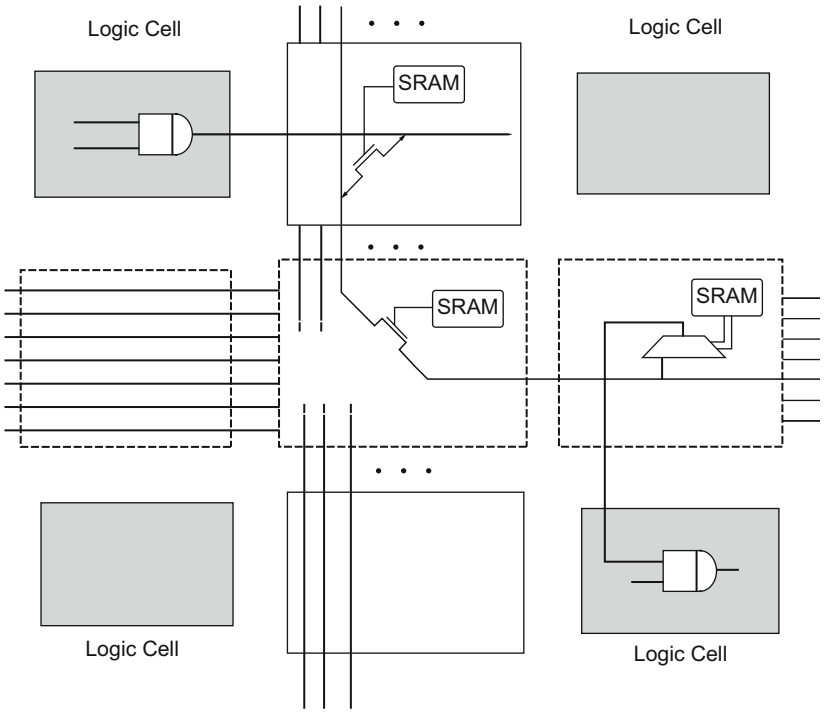


FIGURE 6.18

SRAM-controlled programmable switches.

4. Reduced reliability.
5. Increased cost.

Because of these influences, ASIC packaging has received considerable research and development attention particularly hybrid and multiple-chip module packaging techniques. This sections will only highlight a few of the relevant issues. The interested reader is referred to the professional handbooks devoted to this subject.

The traditional low-cost package of choice has been the dual-inline package. This package has a row of pins on each side, which are mounted into holes drilled through a printed-circuit board. Commercial dual-inline packages grow quite large as pin-count goes up, and are generally limited to a maximum of 64 pins. The pin-grid-array package was developed to increase pin density by placing leads under the entire bottom surface of the package. This technology provides well in excess of 300 pins per package, and like dual-inline packages requires through-hole printed-circuit board mounting with 100 mm lead spacing.

Surface-mount ASIC packages have now overtaken the traditional through-hole package market, even for cost-sensitive products. Here, a chip carrier, which may have leads on all four sides, is soldered directly onto pads on the surface of the printed-circuit board. Lead pitch on a surface mount component

is typically 20 to 50 mils, compared to the 100 mil pitch of dual-inline packages and pin-grid arrays. Higher system packaging density is possible since the ASIC packages are smaller, through-holes are not needed, and components can be placed on both sides of the board without interference. This higher component density reduces parasitic capacitance and inductance and results in higher system operating speed. However, testing of these boards is much more difficult than through-hole printed-circuit boards. For example, traditional bed-of-nails-style board-level testers cannot be used to drive and observe signals from the back side of the board since all ASIC pins are not available on through-holes. In fact, the increasing use of surface-mount printed-circuit boards was a driving factor in the development and overwhelming acceptance of the IEEE Boundary Scan test standard (IEEE 1149.1).

The most promising new ASIC packaging technology is the ball-grid array. A ball-grid-array package provides high I/O density through its array of solder bumps on the underside of the package without requiring ultra-fine-pitch connections to the printed-circuit board. for example, a 1-inch-square quad-flat-packs package with a 50 mil lead pitch can provide 80 I/O connections. For the same package dimensions, a ball-grid-array package can provide 400 I/O connections. They have proved considerable interest since their introduction a few years ago. Motorola is developing a ball-grid-array package for its microcontrollers, Hitachi has planned to offer a micro-ball-grid-array package for its 0.5m mask-pin-grid arrays with up to 672 I/O connections, and Sandia National Labs is developing a mini-ball-grid-array package only slightly larger than the chip die which can accommodate more than 200 I/O connections.

A fundamental constraint imposed by ASIC packaging is the limited number of available pins. During the time when the on-chip ASIC gate count has increased by nearly six orders of magnitude, the number of available package pins has only increased by about two orders of magnitude. The most popular ASIC packages today are the pin-grid arrays, quad-flat-packs and thin-quad-flat-packs. The current trend in packaging is toward very tight lead pitches, staggered lead pitches, advanced array packages such as ball-grid array and flip-chip, and non-standard surface mount packages such as tape-automated bonding.

6.3 PERIPHERAL PROGRAMMABLE-LOGIC DEVICES

A programmable peripheral device is designed to perform various interface functions. Such a device can be set up to perform specific functions by writing an instruction (or instructions) in its internal register, called the control register. This function can be changed during execution by writing a new instruction in this register. These devices are flexible, versatile, and economical; they are widely used in microprocessor-based products.

A programmable device, on the other hand, has its functions determined through software instructions. A programmable peripheral device can not only be viewed as a multiple I/O device, but it also performs many other functions, such as time delay, interrupt handling, and graphic user machine interactions, etc. In fact, it consists of many devices on a single chip, interconnected through a common bus. This is a hardware approach through software control to performing the I/O functions, discussed earlier in this chapter.

This section describes five typical programmable peripheral devices: programmable I/O ports; interrupt controller; timer; CMOS (complementary metal-oxide-semiconductor); and DMA (direct memory access).

6.3.1 Programmable peripheral I/O ports

The programmable peripheral interface, especially the 8255 programmable peripheral I/O interface, is a very popular and versatile input and output chip, easily configured to function in several different configurations. The 8255 is used on several ranges of interface cards that plug into an available slot in controllers or computers, allowing the use of both digital input and output.

As illustrated in Figure 6.19, each 8255 programmable peripheral I/O interface has three 8-bit TTL-compatible I/O ports that will allow the control of up to 24 individual outputs or inputs. For example, they can be attached to a robotic device to control movement by use of motors to control motion and switches to detect position, etc.

Addressing ports is different from addressing memory, as ports have port addresses and memory has memory addresses; port address 1234 is different from memory address 1234. The 8255 programmable peripheral I/O interface cards use port addresses and cannot be set to use memory addresses (see Table 6.2). They plug into any available 8- or 16-bit slot (also known as an AT or ISA slot) on the motherboard of a controller in the same way a sound card or disk drive controller card. The CPU of the motherboard communicates with the cards via its address. By physically using jumpers on the card, we can assign a set of addresses to the card; then in software, we can tell the CPU what these addresses are (more discussion about this are in the FPGA subsection on programming).

Before the chip can be used, its configuration must be set. This tells the 8255 whether ports are input, output or some strange arrangements called bidirectional and strobed. The 8255 allows for three distinct operating modes (modes 0, 1, and 2) as follows:

1. Mode 0: basic input/output. Ports A and B operate as either inputs or outputs and Port C is divided into two 4-bit groups either of which can be operated as inputs or outputs.

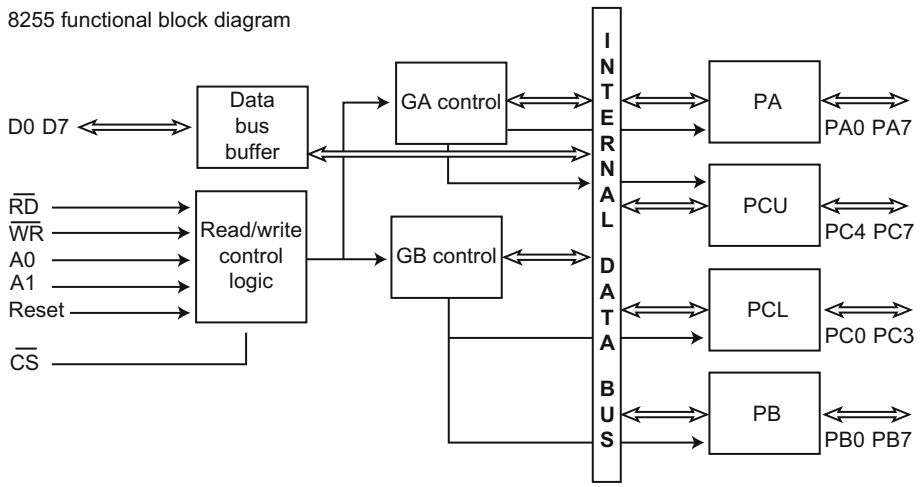


FIGURE 6.19

8255 programmable peripheral I/O interface functional blocks.

Table 6.2 DC-0600 Addresses

Options Address	Option 1: default (JP2 Linked)	Option 2 (JP2 Open)
8255 8255Port	Address [Hex (decimal)]	Address [Hex (decimal)]
Port 1A	300H (768)	360H (864)
Port 1B	301H (769)	361H (865)
Port 1C	302H (770)	362H (866)
Port 1 Control Register	303H (771)	363H (867)
Port 2A	304H (772)	364H (868)
Port 2B	305H (773)	365H (869)
Port 2C	306H (774)	366H (870)
Port 2 Control Register	307H (775)	367H (871)

2. Mode 1: strobed input/output. Same as Mode 0 but Port C is used for handshaking and control.
3. Mode 2: bidirectional bus. Port A is bidirectional (both input and output) and Port C is used for handshaking. Port B is not used.

Mode 0 is the most used. Each of the three ports has 8 bits, and each of these bits can be individually set ON or OFF, somewhat like having three banks of eight light switches. These bits are configured, in groups, to be inputs or outputs. The various modes can be set by sending a value to the control port. The control port is Base Address + 3 (i.e., 768 + 3 = 771 decimal). Table 6.3 shows the different arrangements that can be configured and the values to be sent to the configuration port.

As mentioned, the control port is Base Address + 3. Port A is always at Base Address; Port B is Base Address + 1; Port C is Base Address + 2. Thus, in our example Ports A, B, and C are at 768, 769, and 770 (decimal), respectively. By writing, say, 128, the control port will then configure the 8255 to have all three ports set for output.

Table 6.3 8255 Control Register Configuration (Mode 0)

Control Word [Hex(Dec)]	Port A	Port B
80H (128)	OUT	OUT
82H (130)	OUT	IN
85H (133)	OUT	OUT
87H (135)	OUT	IN
88H (136)	IN	OUT
8AH (138)	IN	IN
8CH (140)	IN	OUT
8FH (143)	IN	IN

6.3.2 Programmable interrupt controller chipset

Both microcontroller and microcomputer system designs require that I/O devices such as keyboards, displays, sensors, and other components receive servicing in an efficient manner so that most of the total system tasks can be assumed by the microcomputer with little or no effect on throughput. As mentioned in subsection 5.1.2, there are two common methods of servicing such devices the polled approach and the interrupt method, which allows the microprocessor to continue executing its main program and only stop to service peripheral devices when told to do so by the device itself.

The programmable interrupt controller (PIC) functions as an overall manager in an interrupt-driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination. The PIC, after issuing an interrupt to the CPU, must input information into the CPU that can "point" the program counter to the correct service routine. This "pointer" is an address in a vectoring table, and will be referred to, in this document, as vectoring data.

The 8259A is taken as an example of the PIC. It manages eight levels of requests and has built-in features for expandability to other 8259As (up to 64 levels). It is programmed by the system's software as an I/O peripheral. A selection of priority modes is available to the programmer so that the manner in which the requests are processed can be configured to match system requirements. Priority modes can be changed or reconfigured dynamically whenever the main program is executing. This means that the complete interrupt structure can be defined on the requirements of the total system environment.

Figure 6.20 gives the block function diagram of 8259A PIC, which includes these function blocks and pins:

1. Interrupt request register (IRR) and in-service register (ISR). The interrupts at the IR input lines are handled by two registers in cascade, the IRR and the ISR. The IRR is used to store all the interrupt levels that are requesting service, and the ISR is used to store all the interrupt levels that are being serviced.
2. Priority resolver. This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during INTA pulse.
3. Interrupt mask register (IMR). The IMR stores the bits that mask the interrupt lines that are to be masked. The IMR operates on the IRR. Masking of a higher-priority input will not affect the interrupt request lines of lower priority.
4. INT (interrupt). This output goes directly to the CPU interrupt input. The VOH level on this line is designed to be fully compatible with the 8080A, 8085A, and 8086 input levels.
5. INTA (interrupt acknowledge). INTA pulses will cause the 8259A to release vectoring information onto the data bus. The format of these data depends on the system mode (mPM) of the 8259A.
6. Data bus buffer. This is a three-state, bidirectional 8-bit buffer that is used to interface the 8259A to the system data bus. Control words and status information are transferred through the data bus buffer.
7. Read/write control logic. The function of this block is to accept OUTPUT commands from the CPU. It contains the initialization command word (ICW) registers and operation command word (OCW) registers that store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the data bus.

8. CS (chip select). A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless the device is selected.
9. WR (write). A LOW on this input enables the CPU to write control words (ICWs and OCWs) to the 8259A.
10. RD (read). A LOW on this input enables the 8259A to send the status of the IRR, ISR, IMR, or the interrupt level onto the data bus.
11. A0. This input signal is used in conjunction with WR and RD signals to write commands into the various command registers, as well as reading the various status-registers of the chip. This line can be tied directly to one of the address lines.
12. The cascade buffer/comparator. This function block stores and compares the IDs of all 8259A that are used in the system. The associated three I/O pins (CAS0-2) are outputs when the 8259A is used as a master, and are inputs when the 8259A is used as a slave. As a master, the 8259A sends the ID of the interrupting slave device onto the CAS0 ± 2 lines. The slave thus selected will send its pre-programmed subroutine address onto the data bus during the next one or two consecutive INTA pulses.

The powerful features of the 8259A in a microcomputer system are its programmability and interrupt routine addressing capability. The latter allows direct or indirect jumping to the specific interrupt routine requested without any polling of the interrupting devices. The normal sequence of events during an interrupt depends on the type of CPU being used. The events occur as follows in an MCS-80/85 system:

1. One or more of the INTERRUPT REQUEST lines (IR7 ± 0) are raised high, setting the corresponding IRR bit(s).
2. The 8259A evaluates these requests and sends an INT to the CPU, if appropriate.

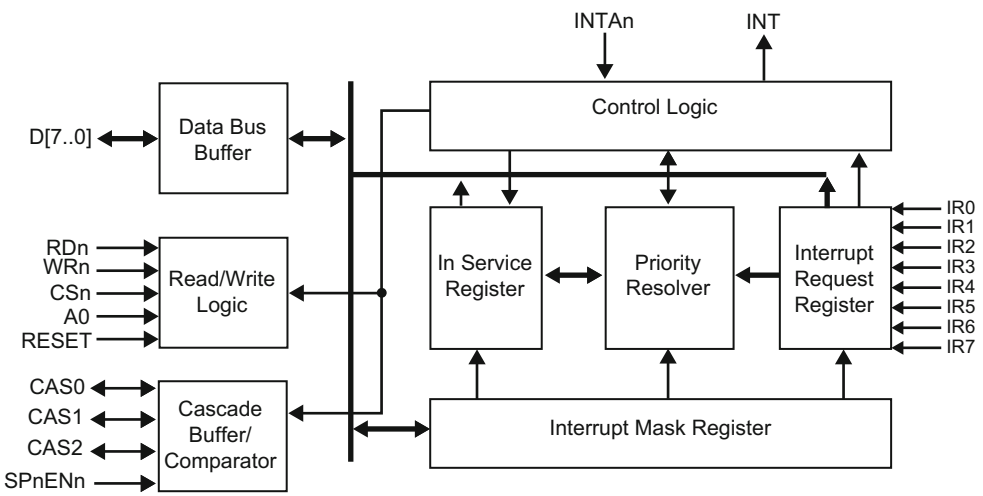


FIGURE 6.20

8259A programmable interrupt controller block diagram.

3. The CPU acknowledges the INT and responds with an INTA pulse.
4. Upon receiving an INTA from the CPU group, the highest-priority ISR bit is set, and the corresponding IRR bit is reset. The 8259A will also release a CALL instruction code (11001101) onto the 8-bit data bus through its D7 ± 0 pins.
5. This CALL instruction will initiate two more INTA pulses to be sent to the 8259A from the CPU group.
6. These two INTA pulses allow the 8259A to release its pre-programmed subroutine address onto the data bus. The lower 8-bit address is released at the first INTA pulse and the higher 8-bit address is released at the second INTA pulse.
7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI mode the ISR bit is reset at the end of the third INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence.

The events occurring in an 8086 system are the same until the fourth step; from the fourth step onward:

(d) Upon receiving an INTA from the CPU group, the highest-priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive the data bus during this cycle.
(e) The 8086 will initiate a second INTA pulse. During this pulse, the 8259A releases an 8-bit pointer onto the data bus where it is read by the CPU.
(f) This completes the interrupt cycle.

In the AEOI mode the ISR bit is reset at the end of the second INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt subroutine. If no interrupt request is present at step (d) of either sequence (i.e., the request was too short in duration) the 8259A will issue an interrupt level 7. Both the vectoring bytes and the CAS lines will look as if an interrupt level 7 was requested.

When the 8259A PIC receives an interrupt, INT becomes active and an interrupt acknowledge cycle is started. If a higher-priority interrupt occurs between the two INTA pulses, the INT line goes inactive immediately after the second INTA pulse. After an unspecified amount of time the INT line is activated again to signify the higher-priority interrupt waiting for service. This inactive time is not specified and can vary between parts. The designer should be aware of this consideration when designing a system that uses the 8259A. It is recommended that proper asynchronous design techniques be followed.

Advanced programmable interrupt controllers (APICs) are designed to attempt to solve interrupt routing efficiency issues in multiprocessor computer systems. There are two components in the Intel APIC system; the local APIC (LAPIC) and the IOAPIC. The LAPIC is integrated into each CPU in the system, and the IOAPIC is used throughout the system's peripheral buses, typically one for each bus. In the original system designs, LAPICs and IOAPICs were connected by a dedicated APIC bus. Newer systems use the system bus for communication between all APIC components.

LAPICs manage all external interrupts for the processor that it is part of. In addition, they are able to accept and generate interprocessor interrupts (IPIs) between LAPICs. LAPICs may support up to 224 usable IRQ vectors from an IOAPIC. Vectors numbers 0 31, out of 0 255, are reserved for exception handling by x86 processors.

IOAPICs contain a redirection table, which is used to route the interrupts it receives from peripheral buses to one or more LAPICs.

6.3.3 Programmable timer controller chipset

The programmable timer controller provides a programmable interval timer and counter that are designed to solve one of the most common problems in any microcomputer system; the generation of accurate time delays by software control. Instead of setting up timing loops in software, the programmer configures the programmable timer controller to the desired delay. After this time, the programmable timer controller will interrupt the CPU. Software overhead is minimal and variable-length delays can easily be accommodated.

Some of the other computer and timer functions that it can implement are real-time clock, event counter, digital one-shot, programmable rate generator, square-wave generator, binary rate multiplier, complex waveform generator, and complex motor controller.

Figure 6.21 gives the typical function blocks for an 82C54 programmable interval timer controller, which has these main blocks:

1. Data bus buffer. This three-state, bidirectional 8-bit buffer is used to interface the 82C54 to the system bus.
2. Read/write logic. The read/write logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 82C54. A1 and A0 select one of the three counters on the control word register to be read from or written into. A "low" on the RD input tells the 82C54 that the CPU is reading one of the counters. A "low" on the WR input tells the 82C54 that the CPU is writing either a control word or an initial count. Both RD and WR are qualified by CS; RD and WR are ignored unless the 82C54 has been selected by holding CS low.
3. Control word register. The control word register is selected by the read/write logic when A1, A0 = 11. If the CPU then does a write operation to the 82C54, the data are stored in the control word

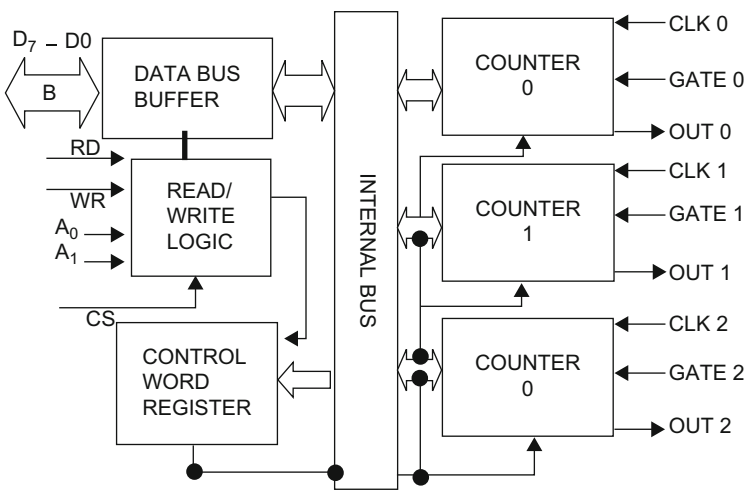


FIGURE 6.21

82C54 programmable timer controller function blocks.

register and interpreted as a control word used to define the counter operation. The control word register can only be written to.

4. Counter 0, Counter 1, Counter 2. These three functional blocks are identical in operation, so only a single counter will be described. The counters are fully independent. Each counter may operate in a different mode.

The programmable timer is normally treated by the system software as an array of peripheral I/O ports; three are counters and the fourth is a control register for mode programming. Basically, the select inputs A0, A1 connects to the A0, A1 address bus signals of the CPU. The CS can be derived directly from the address bus by using a linear select method, or it can be connected to the output of a decoder.

After power-up, the state of the programmable timer is undefined, as are the mode, count value, and output of all counters. Each counter operates a determined when programmed which must happen before use. Unused counters need not be programmed. They are programmed by writing a control word and then an initial count. All control words are written into the control word register, which is selected when A1, A0 = 11. The control word specifies which counter is being programmed. By contrast, initial counts are written into the counters, not the control word register. The A1, A0 inputs are used to select the counter to be written into. The format of the initial count is determined by the control word used.

1. Write operations. A new initial count may be written to a counter at any time without affecting the counter's programmed mode in any way. Counting will be affected as described in the mode definitions.

The new count must follow the programmed count format. If a counter is programmed to read and write 2-byte counts, the following precaution applies. A program must not transfer control between writing the first and second byte to another routine that also writes into that same counter. Otherwise, the counter will be loaded with an incorrect count.

2. Read operations. There are three possible methods for reading the counters. The first is through the read-back command. The second is a simple read operation of the counter, which is selected with the A1, A0 inputs. The only requirement is that the CLK input of the selected counter must be inhibited by using either the GATE input or external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result.

6.3.4 CMOS chipset

The complementary metal-oxide-semiconductor (CMOS) chip is battery-powered and stores the hard drive's configuration and other information. In a microcomputer and a microcontroller, CMOS chips normally provide real-time clock (RTC) and CMOS memory.

The real-time clock provides the board with a time-of-day clock, periodic interrupt, and system configuration information. In personal computers, the CMOS chipset typically contains 64 (00hex-3Fhex) 8-bit locations of battery-backed-up CMOS RAM (random access memory). The split is (1) 00hex 0Ehex, used for real-time clock functions (time of day), (2) 0Fhex 35hex, used for system configuration information, for example, hard drive type, memory size, etc., and (3) 36hex 3Fhex, used for power-on password storage.

CMOS memory is an accessible set of memory locations on the same chip as the RTC and has its own battery backup so that it retains both functions, even when the computer is turned off. Battery-powered CMOS and RTCs did not exist in early systems and the current time was entered manually every time the

system was turned on. This memory location in CMOS is separate from the RTC registers and is generally updated by the BIOS which can update the century information, as can many operating systems, network time systems, and applications, or the user can set it using the right commands.

6.3.5 Direct memory access controller chipset

Direct memory access (DMA) is an I/O technique commonly used for high-speed data transfer among internal memories, I/O ports, and peripherals, and also between the memories and I/O devices on different chipsets. The DMA technique allows the microprocessor to release bus control to a device called a DMA controller. The DMA controller manages data transfer between memory and a peripheral under its control, thus bypassing the microprocessor. The microprocessor communicates with the controller by using the chip select line, buses, and control signals. However, once the controller has gained control, it plays the role of a microprocessor for data transfer. For all practical purposes, the DMA controller is a microprocessor capable only of copying data at high speed from one location to another. An illustration of a programmable DMA controller, the Intel 8237A, is described below.

The 8237A block diagram shown in Figure 6.22 includes its major logic blocks and all the internal registers. The data interconnection paths are also shown. Not shown are the various control signals between the blocks. The 8237A contains 344 bits of internal memory in the form of registers. Table 6.4 lists these registers by name and shows the size of each. It contains three basic blocks of control logic; the timing control block generates internal timing and external control signals for the 8237A. The program command control block decodes the various commands given to the 8237A by the microprocessor prior to servicing a DMA request, and also decodes the mode control word used to select the type of DMA during the servicing. The priority encoder block resolves priority contention between DMA channels that are requesting service simultaneously.

To perform block moves of data from one memory address space to another with a minimum of program effort and time, the 8237A includes a memory-to-memory transfer feature. Programming a bit in the command register selects channels 0 and 1 to operate as memory-to-memory transfer channels. The transfer is initiated by setting the software DREQ for channel 0. The 8237A requests a DMA service in the normal manner. After HLDA is true, the device, using four-state transfers in block transfer mode, reads data from the memory. The channel 0 current address register is the source for the address used and is decremented or incremented in the normal manner. The data byte read from the memory is stored in the 8237A internal temporary register. Channel 1 then performs a four-state transfer of the data from the temporary register to memory using the address in its current address register and incrementing or decrementing it in the normal manner. The channel 1 current word count is decremented. When the word count of channel 1 goes to FFFFH, a TC is generated causing an EOP output terminating the service. Channel 0 may be programmed to retain the same address for all transfers. This allows a single word to be written to a block of memory. The 8237A will respond to external EOP signals during memory-to-memory transfers. Data comparators in block search schemes may use this input to terminate the service when a match is found.

The 8237A will accept programming from the host processor at any time that the HLDA is inactive; this is true even if HRQ is active. The responsibility of the host is to ensure that programming and HLDA are mutually exclusive. Note that a problem can occur if a DMA request occurs on an unmasked channel while the 8237A is being programmed. For instance, the CPU may be starting to reprogram the 2-byte address register of channel 1 when channel 1 receives a DMA request. If the 8237A is

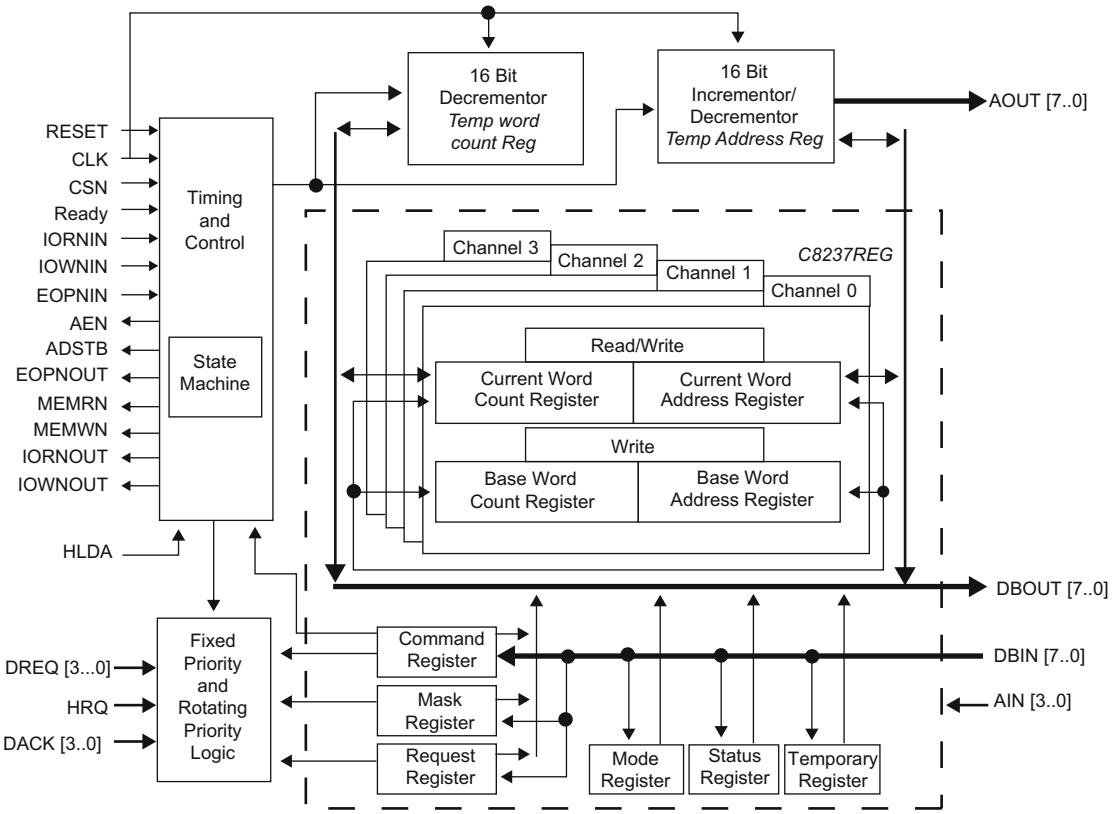


FIGURE 6.22

Intel 8237A DMA controller block diagram.

enabled (bit 2 in the command register is 0) and channel 1 is unmasked, a DMA service will occur after only 1 byte of the address register has been reprogrammed. This can be avoided by disabling the controller (setting bit 2 in the command register) or masking the channel before programming any other registers. Once the programming is complete, the controller can be enabled and unmasked. After power-up it is suggested that all internal locations, especially the mode register, be loaded with some valid value. This should be done even if some channels are unused. An invalid mode may force all control signals to go active at the same time.

The 8237A is designed to operate in two major cycles. These are called idle and active cycles. Each device cycle is made up of a number of states. The 8237A can assume seven separate states, each composed of one full clock period. State I (SI) is the inactive state. It is entered when the 8237A has no valid DMA requests pending. While in SI, the DMA controller is inactive but may be in program condition, being programmed by the processor. State S0 (S0) is the first state of a DMA service. The 8237A has requested a hold but the processor has not yet returned an acknowledgement. The 8237A may still be programmed until it receives HLDA from the CPU. An acknowledgement from the CPU

Table 6.4 8237A DMA Controller Internal Registers

Name	Size	Number
Base address registers	16 bits	4
Base word count registers	16 bits	4
Current address registers	16 bits	4
Current word count registers	16 bits	4
Temporary address register	16 bits	1
Temporary word count register	16 bits	1
Status register	8 bits	1
Command register	8 bits	1
Temporary register	8 bits	1
Mode registers	6 bits	4
Mask register	4 bits	1
Request register	4 bits	1

will signal that DMA transfers may begin. S1, S2, S3, and S4 are the working states of the DMA service. If more time is needed to complete a transfer than is available with normal timing, wait states (SW) can be inserted between S2 or S3 and S4 by the use of the ready line on the 8237A. Eight states are required for a single transfer. The first four states (S11, S12, S13, S14) are used for the read-from-memory half and the last four states (S21, S22, S23, S24) for the write-to-memory half of the transfer.

(1) Idle cycle

When no channel is requesting service, the 8237A will enter the idle cycle and perform "SI" states. In this cycle the 8237A will sample the DREQ lines every clock cycle to determine whether any channel is requesting a DMA service. The device will also sample CS, looking for an attempt by the microprocessor to write or read the internal registers of the 8237A. When CS is low and HLDA is low, the 8237A enters the program condition. The CPU can now establish, change, or inspect the internal definition of the part by reading from or writing to the internal registers. Address lines A0 $\pm$ A3 are inputs to the device and select which registers will be read or written. The IOR and IOW lines are used to select and time reads or writes. Special software commands can be executed by the 8237A in the program condition. These commands are decoded as sets of addresses with the CS and IOW. The commands do not make use of the data bus. Instructions include Clear First/Last Flip-Flop and Master Clear.

(2) Active cycle

When the 8237A is in the idle cycle and a nonmasked channel requests a DMA service, the device will output an HRQ to the microprocessor and enter the active cycle. It is in this cycle that the DMA service will take place, in one of four modes:

1. Single transfer mode. In single transfer mode the device is programmed to make one transfer only. The word count will be decremented, and the address decremented or incremented following each transfer. When the word count rolls over from 0 to FFFFH, a terminal count (TC) will cause an

auto-initialize if the channel has been programmed to do so. DREQ must be held active until DACK becomes active in order to be recognized. If DREQ is held active throughout the single transfer, HRQ will go inactive and release the bus to the system. It will again go active and, upon receipt of a new HLDA, another single transfer will be performed. Details of timing between the 8237A and other bus control protocols will depend upon the characteristics of the microprocessor involved.

2. Block transfer mode. In block transfer mode the device is activated by DREQ to continue making transfers during the service until a TC, caused by word count going to FFFFH, or an external end-of-process (EOP) is encountered. DREQ need only be held active until DACK becomes active. Again, an auto-initialization will occur at the end of the service if the channel has been programmed for it.
3. Demand transfer mode. In demand transfer mode the device is programmed to continue making transfers until a TC or external EOP is encountered or until DREQ goes inactive. Thus, transfers may continue until the I/O device has exhausted its data capacity. After the I/O device has had a chance to catch up, the DMA service is re-established by means of a DREQ. During the time between services when the microprocessor is allowed to operate, the intermediate values of address and word count are stored in the 8237A current address and current word count registers. Only an EOP can cause an auto-initialize at the end of the service. EOP is generated either by TC or by an external signal. DREQ has to be low before S4 to prevent another transfer.
4. Cascade mode. This mode is used to cascade more than one 8237A together for simple system expansion. The HRQ and HLDA signals from the additional 8237A are connected to the DREQ and DACK signals of a channel of the initial 8237A. This allows the DMA requests of the additional device to propagate through the priority network circuitry of the preceding device. The priority chain is preserved and the new device must wait for its turn to acknowledge requests. Since the cascade channel of the initial 8237A is used only for prioritizing the additional device, it does not output any address or control signals of its own. These could conflict with the outputs of the active channel in the added device. The 8237A will respond to DREQ and DACK but all other outputs except HRQ will be disabled. The ready input is ignored.

Each of the three active transfer modes above can perform three different types of transfers; read, write, and verify. Write transfers move data from an I/O device to the memory by activating MEMW and IOR. Read transfers move data from memory to an I/O device by activating MEMR and IOW. Verify transfers are pseudotransfers. The 8237A operates as in read or write transfers generating addresses, and responding to EOP, etc. However, the memory and I/O control lines all remain inactive. The ready input is ignored in verify mode.

Problems

1. Please identify which of these two programmable logic and application specific IC designs, system on chip and network on chip, should be an option for ASIC design?
2. Please name all primitive parts in the ASIC fabrication technologies.
3. Those ASICs made for CMOS are manufactured by a process of repeatedly etching a chemical resist layer on a silicon wafer to a required pattern, and then chemically diffusing, growing or etching layers. Please state how many layers are implemented on a silicon wafer in the ASIC fabrication process.

4. How would you define standard cell ASICs?
5. Based on Figure 6.4, write an abstraction of the standard cell design flow (steps) given in subsection 6.1.3 of this textbook.
6. Based on the example given in Figure 6.8, explain the programming principle for the simple logic module in FPGAs.
7. Based on the example given in Figure 6.10, explain the programming principle for the combinatorial logic module in FPGAs.
8. Based on Figure 6.12 and Figure 6.13, explain the programming principle for the adaptive logic module in FPGA.
9. Do some website research and name several SPLD products available on the market.
10. Do some website research and name several CPLD products available on the market.
11. Why does one 8255 Programmable peripheral I/O port exactly allow the control of up to 24 individual outputs, or up to 24 individual inputs, or up to 24 inputs and outputs?
12. Based on Figure 6.20, explain the working principle of 8259A PIC, and explain why 8259A PIC can enhance the interrupt routing efficiency.
13. Please give one or two primitives/semantics in a real time operating system which need to set an accurate time delay; then explain how a programmable timer chip can help the microprocessor to implement this function.
14. What similarities and differences exist between programmable I/O ports and a DMA controller?

Further Reading

Dave Landis. Programmable Logic and Application Specific Integrated Circuits. Chapter II in Handbook of Components for Electronics. McGraw Hill. 1998.

Southeast University (www.seu.edu.cn). ASICs the website. http://iroi.seu.edu.cn/books/asics/ASICs.htm#anchor11320. Accessed: February 2009.

Paul Naish. Designing Asics. http://web.ukonline.co.uk/paul.naish/DA/ch1.htm. Accessed: February 2009.

Wikipedia (http://en.wikipedia.org). Standard cell. http://en.wikipedia.org/wiki/Standard cell. Accessed: February 2009.

Tutorial Report (http://www.tutorial reports.com). ASIC design guide. http://www.tutorial reports.com/hardware/asic/tutorial.php. Accessed: May 2007.

Tutorial Report (http://www.tutorial reports.com). ASIC simulation and synthesis. http://www.tutorial reports.com/hardware/asic/synthesis.php. Accessed: May 2007.

Tutorial Report (http://www.tutorial reports.com). FPGA tutorials. http://www.tutorial reports.com/computerscience/fpga/tutorial.php. Accessed: May 2007.

Ian Kuon et al. FPGA Architecture: Survey and Challenges. Now Publishers. 2008.

Altera Corporation (http://www.altera.com). FPGA architecture: WP 01003 1.0. July 2006.

Stephen Brown and Jonathan Rose. Architecture of FPGAs and CPLDs: a tutorial. http://www.eecg.toronto.edu/~jayar/pubs/brown/survey.pdf. Accessed: February 2009.

Actel Corporation (http://www.actel.com). Introduction to Actel FPGA architecture: Actel Application Note AC165. 2000.

David V. Anderson, Paul E. Hasler. FPGA key technologies. http://cadsp.ece.gatech.edu/keytech.shtml. Accessed: February 2009.

Pdf Search Engine (http://www.pdf search engine.com). FPGA PDF. http://www.pdf search engine.com/fpga architecture pdf.html. Accessed: February 2009.

Intel (http://www.intel.com). http://www.intel.com/pressroom/kits/quickreffam.htm. Accessed: June 2007.

Intel (http://www.intel.com). 8259A programmable interrupt controller. http://bochs.sourceforge.net/techspec/intel 8259a pic.pdf.gz. Accessed: June 2007.

Intel (http://www.intel.com). 82C54 programmable interval timer. http://bochs.sourceforge.net/techspec/intel 82c54 timer.pdf.gz. Accessed: June 2007.

Intel (http://www.intel.com). 8237A programmable DMA controller. http://bochs.sourceforge.net/techspec/intel 8237a dma.pdf.gz. Accessed: June 2007.

MITRE (http://www.mitre.org). Real Time Clock and CMOS. http://www.mitre.org/tech/cots/RTC.html. Accessed: December 2008.

Netrino (http://www.netrino.com). SPLD CPLD PLD. http://www.netrino.com/Publications/index.php. Accessed: December 2008.

Patterson, David A. and Hennessey L. John. Computer Organization and Design. Second Edition. San Francisco: Morgan Kaufmann. 1998.

CHAPTER

Industrial intelligent controllers 7

Intelligent control describes the discipline where control methods are developed that attempt to emulate important characteristics of human intelligence. These characteristics include adaptation and learning, planning under large uncertainty and coping with large amounts of data. Accordingly, the term "intelligent control" has come to mean some form of control using fuzzy logic, digital control algorithms, numerical mathematical computations, and supervisory strategies, when applied to large instrument networks and large data pools. Intelligent control, however, does not restrict itself only to those logic theories, computational methods, and control strategies. Today, it control tends to encompass everything not characterized as conventional control; it has, however, shifting boundaries and what is called intelligent control today will probably be called conventional control tomorrow.

From a systems point of view, the use of intelligent control is a natural next step in the quest to control complex industrial process and production systems since there are needs today that cannot be successfully addressed by conventional control theory. These mainly pertain to areas where heuristic methods may be needed to improve a control law or where novel control functions need to be developed, whilst the system remains in operation. Learning from past experience and planning control actions may be necessary, as may failure detection and identification who use Such functions have been performed in the past by human operators so, when high-level decision-making techniques for reasoning under uncertainty these techniques are used by humans, attributed to intelligent behavior. Hence, one way to achieve a high degree of automation is to utilize high-level decision-making techniques and intelligent methods in the autonomous controller.

To increase the speed of response, to relieve the operators from mundane tasks, to protect them from hazards, a high degree of autonomy is desired. Automation is the objective, and intelligent controllers are one way to achieve it.

7.1 PLC (PROGRAMMABLE LOGIC CONTROL) CONTROLLERS

The development of programmable logic controllers (PLCs) was driven primarily by the requirements of automobile manufacturers, who constantly changed their production line control systems to accommodate their new car models. In the past, this required extensive rewiring of banks of relays a very expensive procedure. In the 1970s, with the emergence of solid-state electronic logic devices, several automobile companies challenged control manufacturers to develop a means of changing control logic without the need to totally rewire the system. The PLC evolved from this requirement. PLCs are designed to be relatively user-friendly so that electricians can easily make the transition from all-relay control to electronic systems. They give users the capability of displaying and troubleshooting ladder-logic that shows the logic in real time. The ladder-logic can be "rewired" (programmed) and tested, without the need to assemble and rewire banks of relays.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10007-5

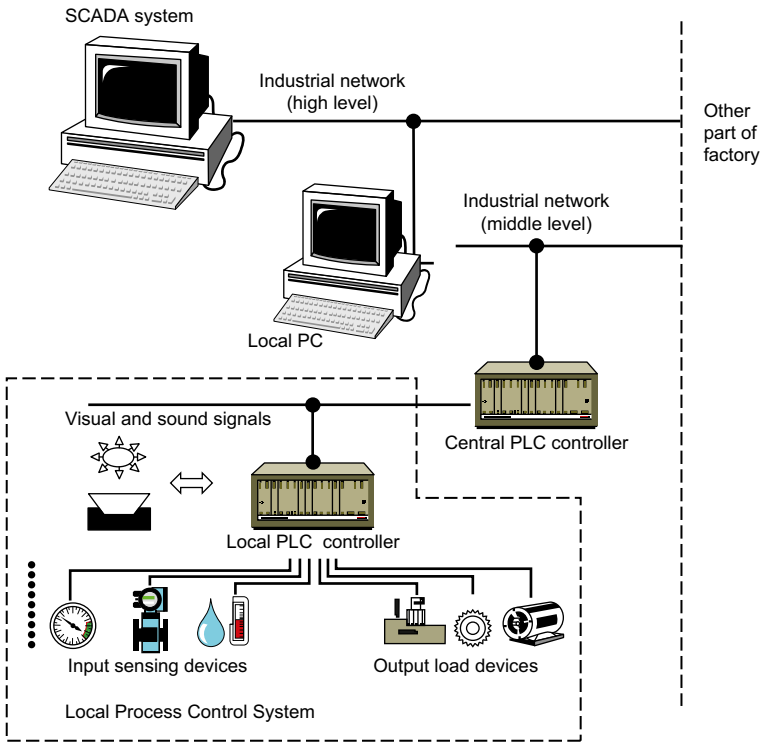


FIGURE 7.1

Schematic of the PLC control network.

A PLC is a computer with a single mission. It usually lacks a monitor, a keyboard, and a mouse, as normally it is programmed to operate a machine or a system using only one program. The machine or system user rarely, if ever, interacts directly with the program in the PLC. When it is necessary to either edit or create the PLC program, a personal computer is usually (but not always) connected to it. The information from the PLCs can be accessed by supervisory control and data acquisition (SCADA) systems and human machine interfaces (HMIs), to provide a graphical representation of the status of the plant. Figure 7.1 is a schematic of the PLC control network resident in industrial systems.

7.1.1 PLC components and architectures

A PLC is actually an industrial microcontroller system (in more recent years we meet microprocessors instead of microcontrollers) with hardware and software that are specifically adapted to the industrial environment. A block schema with the typical components that a PLC consists of is shown in Figure 7.2. Special attention needs to be given to input and output, because most PLC models feature a vast assortment of interchangeable I/O modules that allow for convenient interfacing with virtually

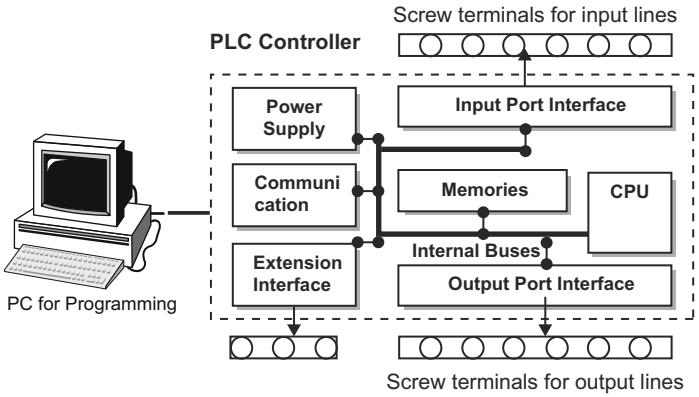


FIGURE 7.2

Basic elements of a PLC controller.

any kind of industrial or laboratory equipment. The program unit is usually a computer used for writing a program (often in a ladder-logic diagram).

(1) Central processing unit (CPU)

This unit contains the "brains" of the PLC, often referred to as a microprocessor or scheduler. The basic instruction set is a high-level program, installed in read-only memory (ROM). The programmed logics are usually stored in electrically erasable permanent read-only memory (EEPROM). The CPU will save everything in memory, even after a power loss. Since it is "electrically erasable", the logic can be edited or changed as the need arises. The programming device is connected to the CPU whenever the operator needs to monitor, troubleshoot, edit, or program the system, but it is not required during normal operations.

(2) Memory

System memory (today mostly implemented in Flash technology) is used by a PLC as a process control system. Aside from this operating system, it also contains a user program translated from a ladder-logic diagram to binary form. Flash memory contents can be changed only when the user program is being changed. PLC controllers were used before Flash memory, with EPROM instead of Flash memory which needs to be erased with a UV lamp and programmed on programmers. The use of Flash technology has greatly shortened this process and allows reprogramming via a serial cable in an application development program.

User memory is divided into blocks, which have special functions. Some parts are used for storing input and output status. The real status of an input is stored either as "1" or as "0" in a specific memory bit. Each input or output has one corresponding bit in memory. Other parts of the memory are used to store variables in user programs, for example, timer value or counter value.

PLC controller memory consists of several areas, as given in Table 7.1, some of which have predefined functions.

Table 7.1 Memory Structure of a PLC

Data area		Word(s)	Bit(s)	Function
IR area	Input area	IR 000 – IR 009 (10 words)	IR 00000 – IR 00915 (160 bits)	These bits may be assigned to an external I/O connection. Some of these have direct output on a screw terminal (for example, IR000.00 - IR000.05 and IR010.00 - IR010.03 with CPM1A model)
	Output area	IR 010 – IR 019 (10 words)	IR 01000 – IR 01915 (160 bits)	
	Working area	IR 200 – IR 231 (32 words)	IR 20000 – IR 23115 (512 bits)	Working bits that can be used freely in the program. They are commonly used as swap bits
SR area		SR 232 – SR 255 (24 words)	SR23200 – SR25515 (384 bits)	Special functions, such as flags and control bits
TR area		—	TR 0 – TR 7 (8 bits)	Temporary storage of ON/OFF states when jump takes place
HR area		HR 00 – HR 19 (20 words)	HR0000 – HR1915 (320 bits)	Data storage; these keep their states when power is off
AR area		AR 00 – AR 15 (16 words)	AR0000 – AR1515 (256 bits)	Special functions, such as flags and control bits
LR area		LR 00 – LR 15 (16 words)	LR0000 – LR1515 (256 bits)	1:1 connection with another PC
Timer/counter area		TC 000 – TC 127 (timer/counter numbers)		Same numbers are used for both timers and counters
DM area	Read/ write	DM 0000 - DM 0999 and DM 1022 - DM 1023 (1002 words)	—	Data of DM area may be accessed only in word form. Words keep their contents after the power is off
	Error writing	DM 1000 - DM 1021 (22 words)	—	Part of the memory for storing the time and code of error that occurred. When not used for this purpose, they can be used as regular DM words for reading and writing. They cannot be changed from within the program
	Read only	DM 6144 - DM 6599 (456 words)	—	
	PC setup	DM 6600 - DM 6655 (56 words)	—	Storing various parameters for controlling the PC

Note:

1. IR and LR bits, when not used to their purpose, may be used as working bits.
2. Contents of HR area, LR area, counter and DM area for reading/writing is stored within backup condenser. At 25°C, condenser keeps the memory contents for up to 20 days.
3. When accessing the current value of PV, TC numbers used for data have the form of words. When accessing the Completing flags, they are used as data bits.
4. Data from DM6144 to DM6655 must not be changed from within the program, but can be changed by peripheral device.

(3) Communication board

Every brand of PLC has its own programming hardware, such as a small hand-held device, which resembles an oversized calculator with a liquid crystal display (LCD) but most commonly this is computer-based. Computer-based programmers typically use a special communication board, installed in an industrial terminal or personal computer, which runs the appropriate software program. This allows offline programming, where program logic is developed separately, then loaded onto the CPU when required.

Programming can be done directly into the CPU if desired; in which case, a programmer can test the system, and watch the logic operate as each element is highlighted in sequence on a cathode ray tube (CRT) when the system is running. Since a PLC can operate without having the programming device attached, one device can be used to service many separate PLC systems.

(4) PLC controller inputs

The intelligence of an automated system depends largely on the ability of a PLC controller to read signals from different types of sensors and input devices. Keys, keyboards, and functional switches are the basis of the human machine relationship. On the other hand, to detect a working piece, view a mechanism in motion, or check pressure or fluid level specific automatic devices such as proximity sensors, marginal switches, photoelectric sensors, level sensors, and so on are needed. Thus, input signals can be logical (ON/OFF) or analog in order to receive input from these devices. Smaller PLC controllers usually have digital input lines only while larger ones also accept analog inputs through special units. One of the most frequent analog signals is a current signal of 4 20 mA and a millivolt voltage signal generated by the various sensors that are usually used as inputs for PLCs. You can obtain sensors for different purposes.

Other devices also can serve as inputs to the PLC controller, such as intelligent devices such as robots, video systems, and so forth (a robot, for instance, can send a signal to PLC controller input as information when it has finished moving an object from one place to the other).

(5) PLC controller output

An industrial control system is incomplete if it is not connected to some output devices. Some of the most frequently used devices are motors, solenoids, relays, indicators and sound emitters. By starting a motor, or a relay, the PLC can manage or control a simple system, such as a system for sorting products, or even complex systems such as a service system for positioning the head of a robotic machine. Output can be analog or digital. A digital output signal works as a switch; it connects and disconnects lines. Analog output is continous, and used to generate an analog signal (for instance, a motor whose speed is controlled by a voltage, the value of which corresponds to a desired speed).

(6) Extension lines

Every PLC controller has a limited number of input/output lines, which can be increased by using extension lines, which can be applied to both input and output lines. Also, extension modules can have inputs and outputs that are different from those on the PLC controller (for instance, if relay outputs are on a controller, transistor outputs can be on an extension module). The PLC has input and output lines through which it is connected to a system it directs.

Two terms frequently mentioned when discussing connections to inputs or outputs are "sinking" and "sourcing"; both are very important in connecting a PLC correctly to the external environment. The briefest definition of these two concepts would be:

$$\text{Sinking} = \text{Common GND line } (-);$$

$$\text{Sourcing} = \text{Common VCC line } (+),$$

Hence the (+) and the (−) in the above definition, which refer to the poles of a DC supply. Inputs and outputs that are either sinking or sourcing can conduct electricity only in one direction, so they are only supplied with direct current. According to what we have discussed so far, each input or output has its own return line, so five inputs would need 10 screw terminals on a PLC controller housing. Instead, we use a system of connecting several inputs to one return line as illustrated in Figure 7.2. These common lines are usually marked "COMM" on the PLC controller housing.

(7) Power supply

Most PLC controllers work either at 24 V DC or 220 V AC. On some PLC controllers, usually the bigger ones, you will find the electrical supply as a separate module while small and medium series already contain the supply module. Different types of modules use different amounts of electrical current so the user determine the correct setting.

This electrical supply is not usually used to start external inputs or outputs. The user has to provide separate supplies in starting PLC controller inputs or outputs to ensure a so-called pure supply for the PLC controller. By pure supply we mean a supply where the industrial environment cannot affect it adversely. Some of the smaller PLC controllers supply their inputs with voltage from a small supply source already incorporated into the PLC.

The internal logic and communication circuitry usually operates on 5 or 15 V DC power. The power supply provides filtering and isolation of the low-voltage power from the AC power line. Power supply assemblies may be separate modules, or in some cases plug-in modules in the I/O racks. Separate control transformers are often used to isolate inputs and CPU from output devices. The purpose is to isolate this sensitive circuitry from transient disturbances produced by any highly inductive output devices.

(8) Timers and counters

Timers and counters are indispensable in PLC programming for numbering products, determining the time for a required action, and so on. Timing functions are very important, and cycle periods are critical in many processes.

There are two types of timers; delay-off and delay-on. The first has a delay before turn off and the second has a delay before turning on in relation to the activation signal. An example of a delay-off timer would be staircase lighting, which simply turns off a few minutes after its activation. Each timer has a time basis, or more precisely has several time bases. Typical values are 1, 0.1, and 0.01 second. If the programmer has entered 0.1 as the time basis and 50 as the delay increase number, the timer will have a delay of 5 seconds (50×0.1 seconds $= 5$ seconds).

Timers also have to have the SV value the number of increments that the timer has to calculate before it changes the output status in advance, either as a constant or a variable. If a variable is

used, the timer will use a real-time value of the variable to determine a delay. This enables delays to vary depending on the conditions of the process. An example is a system that produces two different products, each requiring different timing during the process itself. Product A requires a period of 10 seconds, so number 10 would be assigned to the variable. When product B appears, the variable can change to that required by product B.

Typically, timers have two inputs. The first is the timer-enable, or conditional input (when this input is activated, the timer will start counting), and the second input is a reset input. This input has to be in OFF status in order for a timer to be active. Some PLC models require this input to be low for a timer to be active; other makers require high status (all of them function in the same way basically). However, if a reset line changes status, the timer erases accumulated value.

7.1.2 PLC control mechanism

A programmable logic controller is a digital electronic device that uses programmable memory to store instructions, and a CPU to implement specific functions such as logic, sequence, timing, counting, and arithmetic, in order to control machines and processes. Figure 7.2 shows a simple schematic of a typical programmable logic controller. When running, the CPU scans the memory continuously from top to bottom, and left to right, checking every input, output, and instruction in sequence. The scan time depends upon the size and complexity of the program, and the number and type of I/O, and may be as short as a few milliseconds, so producing tens of scans per second. This short time makes the operation appear as instantaneous, but one must consider the scan sequence when handling critically timed operations and sealing circuits. Complex systems may use interlocked multiple CPUs to minimize total scan time.

The input and output modules allow the PLC to communicate with the machine and are quite different to those in a PC. The inputs may come from limit switches, proximity sensors, temperature sensors, and so on. The PLC will set the outputs on the basis of the software program and the combination of inputs . These outputs may control motor speed and direction, actuate valves, open or close gates, and control all the motions and activities of the machine.

(1) System address

The key to getting comfortable with any PLC is to understand the total addressing system. We have to connect our discrete inputs, push-buttons, limit-switches, and so on, to our controller, interface those points with the “electronic ladder-logic diagram” (program), and then bring the results out through another interface to operate motor starters, solenoids, lights, and so forth.

Inputs and outputs are wired to interface modules, installed in an I/O rack. Each rack has a two-digit address, each slot has its own address, and each terminal point is numbered. Figure 7.3 shows a PLC product in which all of these addresses are octal.

We combine the addresses to form a number that identifies each input and output.

Some manufacturers use decimal addresses, and some older systems are based on 8-bit words, rather than 16. There are a number of proprietary programmable controllers for specific applications, such as elevator controls or energy management, which may not follow the expected pattern, but they will use either 8- or 16-bit word structures. It is very important to identify the addressing system before you attempt to work on any system that uses a programmable controller, because one must know the purpose of each I/O bit before manipulating them in memory.

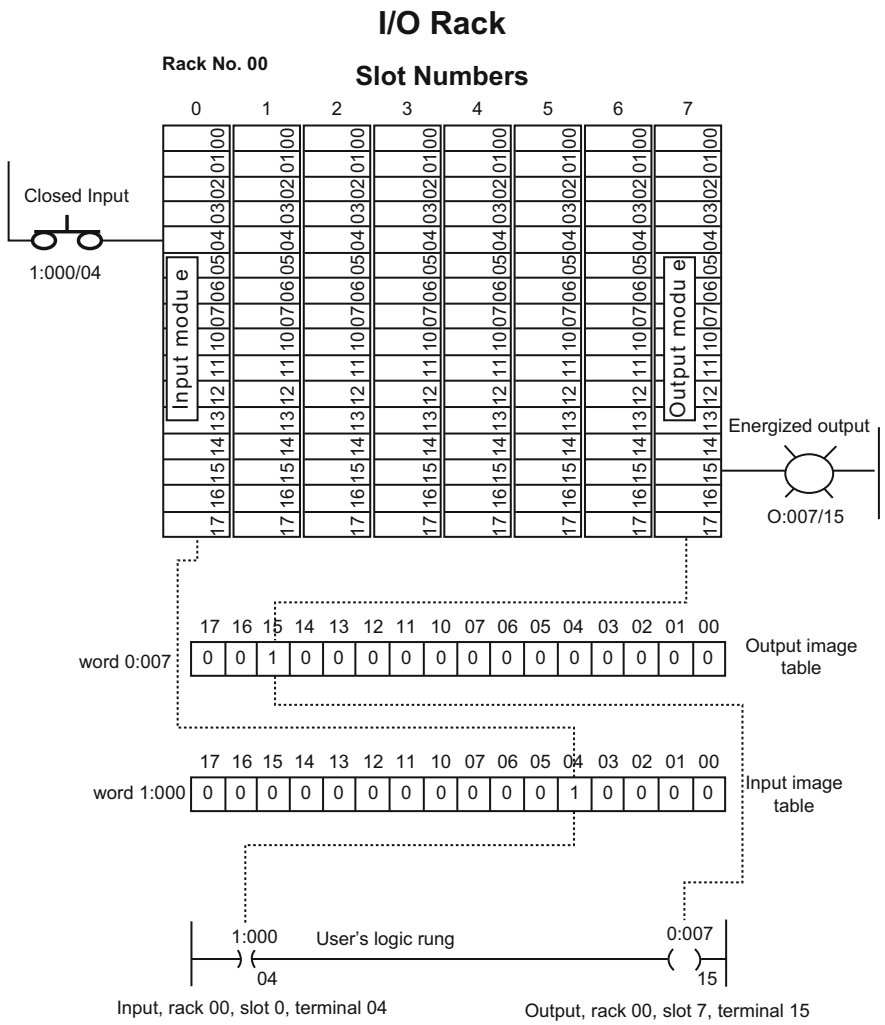


FIGURE 7.3

Solution of one line of logic.

If you know the address of the input or output is known the status of its bit can be checked immediately by calling up the equivalent address on a CRT screen.

(2) I/O addresses

Figure 7.4 gives an I/O address scheme, which shows us that the I/O modules are closely linked with the input and output image tables, respectively.

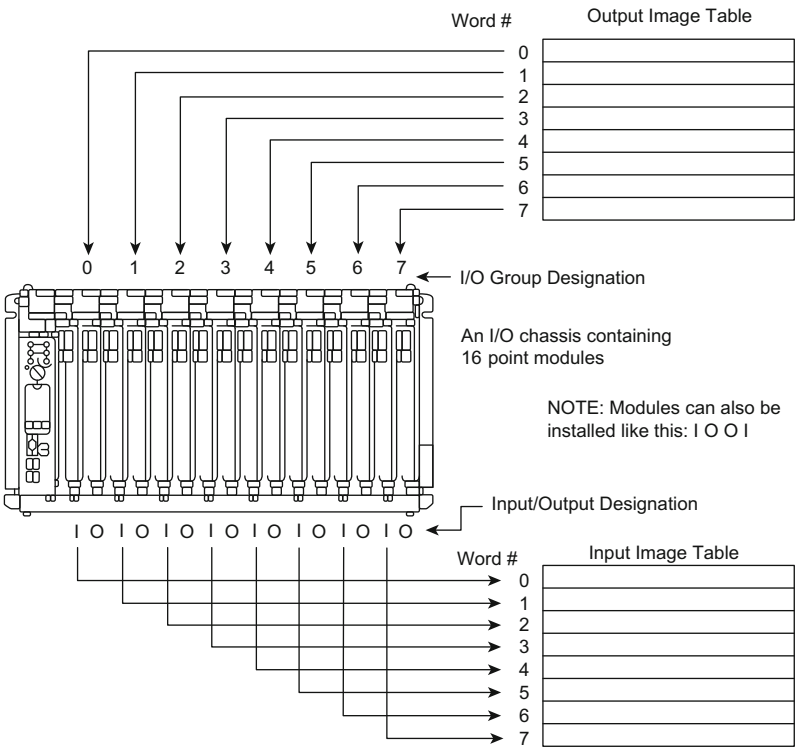


FIGURE 7.4

I/O addressing scheme of a PLC.

Figure 7.3 shows a very simple line of logic, where a push-button is used to turn on a lamp. The push-button and lamp "hardwiring" terminates at I/O terminals, and the logic is carried out in software. We have a push-button, wired to an input module (I), installed in rack 00, slot 0, and terminal 04. The address becomes I:000/04. An indicating lamp is wired to an output module (O), installed in rack 00, slot 7, and terminal 15. The address becomes O:007/15. Our input address, I:000/04, becomes memory address I:000/04, and the output address 0:007/15 becomes memory address 0:007/15.

In other words, the type of module, the rack address, and the slot position identifies the word address in memory. The terminal number identifies the bit number.

(3) Image table addresses

An output image table is reserved in its IR area of the memory (see Table 7.1) as File format, and an input image table is reserved in the same way. A File in memory contains any number of words. Files are separated by type, according to their functions. In the same way, an input image table is also reserved in its IR area of the memory (See Table 7.1) in file format. Figure 7.4 illustrates the respective mapping relationship of the I/O modules to both output and input image tables.

(4) Scanning

As the scan reads the input image table, it notes the condition of every input, and then scans the logic diagram, updating all references to the inputs. After the logic is updated, the scanner resets the output image table, to activate the required outputs. Figure 7.4 shows some optional I/O arrangements and addressing.

In Figure 7.3, we show how one line of logic would perform when the input at I:000/04 is energized: it immediately sets input image table bit I:000/04 true (ON). The scanner senses this change of state, and makes the element I:000/04 true in our logic diagram. Bit 0:007/15 is turned on by the logic. The scanner sets 0:007/15 true in the output image table, and then updates the output 0:007/15 to turn the lamp on.

7.1.3 PLC programming

Programmable logic controllers use a variety of software programming languages. These include sequential function chart (SFC), function block diagram (FBD), ladder-logic diagram (LD), structured text (ST), instruction list (IL), relay ladder logic (RLL), flow chart, C, C++, and Basic. Among these languages, the ladder diagram is the most popular. Almost every language possesses various useful options, such as forced switching on and off of the system inputs/outputs (I/O lines), program follow-up in real time, or documenting a diagram very necessary to understand and to define failures and malfunctions. The programmer can add remarks, names of input or output devices, and comments that can be useful when finding errors, or with system maintenance. Adding comments and remarks enables any technician (and not just the person who developed the system) to understand a ladder-logic diagram easily. Comments and remarks can even precisely quote part numbers in case replacements are needed which speeds up repair of any problems arising because of faulty parts. Formerly, only the person who developed the system had access to the program, so nobody apart from them could understand how it was done. A correctly documented ladder-logic diagram allows any technician to understand thoroughly how the system functions.

(1) Relay ladder logic

Ladder logic, developed to mimic relay logic, is the main programming method used for PLCs. Relays are used to let one power source close a switch for another (often high current) power source, while keeping them isolated. An example of a relay in a simple control application is shown in Figure 7.5. In this system, the first relay (on the left) is normally closed and will allow current to flow until a voltage is applied to input A. The second relay is normally open and will not allow current to flow until a voltage is applied to input B. If current is flowing through the first two relays, then current will flow through the coil in the third relay, and close the switch for output C. This circuit would normally be drawn in the ladder logic form. This can be read logically as C will be on if A is off and B is on.

The example in Figure 7.5 does not show the entire control system, but only its logic. Figure 7.6 shows a more complete representation of the PLC. Here there are two inputs from push-buttons and the input, output and logic are all shown.

We can imagine the inputs as activating 24 V DC relay coils in the PLC. This in turn drives an output relay that switches 115 V AC, which will turn on a light. Note; in actual PLCs inputs are never relays, but often outputs are. The ladder logic in the PLC is actually a computer program that the user can enter and change. Note that both of the input push-buttons are normally open, but the ladder logic

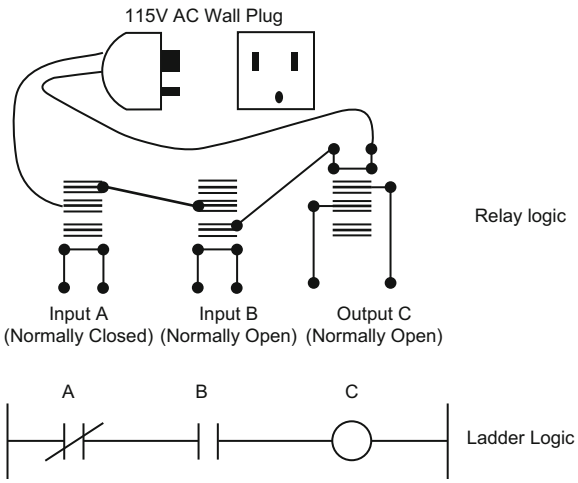


FIGURE 7.5

A simple relay controller.

inside the PLC has one normally open contact and one normally closed. Do not think that the ladder logic in the PLC needs to match the inputs or outputs. Many beginners will get caught trying to make the ladder logic match the input types.

Many relays also have multiple outputs (throws) and this allows an output relay to also be an input simultaneously. The circuit shown in Figure 7.7(a) is an example of this; it is called a seal in circuit. Here, the current can flow through either branch of the circuit, through the contacts labeled A or B. The input B will only be on when the output B is on. If B is off, and A is energized, then B will turn on. If B turns on then the input B will turn on and keep output B on even if input A goes off. After B is turned on the output, B will not turn off.

PLC inputs are easily represented in ladder logic. Figure 7.7(b) shows the three types of inputs. The first two are normally open and closed inputs, discussed previously. Normally open: an active input X will close the contact and allow power to flow. Normally closed: power flows when the input X is not open. The IIT (Immediate Input Terminal) function allows inputs to be read after the input scan, while the ladder logic is being scanned. This allows ladder logic to examine input values more often than once every cycle. Immediate inputs will take current values, but not those from the previous input scan.

In ladder logic, there are multiple types of outputs, but these are not consistently available on all PLCs. Some will be externally connected, as may the internal memory locations in the PLC. Six types of outputs are shown in Figure 7.7(c). The first is a normal output; when energized it will turn on and energize an output. The circle with a diagonal line through it is a normally on output. When it is energized, the output will turn off. This type of output is not available on all PLC types. When initially energized, the OSR (one-shot relay) instruction will turn on for one scan, but then be off for all scans after, until it is turned off. The L (latch) and U (unlatch) instructions can be used to lock outputs on. When an L output is energized, the output will turn on indefinitely, even when the output coil is

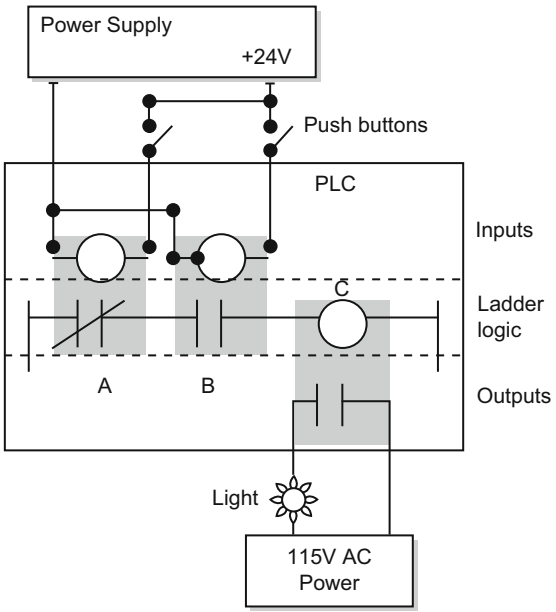


FIGURE 7.6

A simple relay controller.

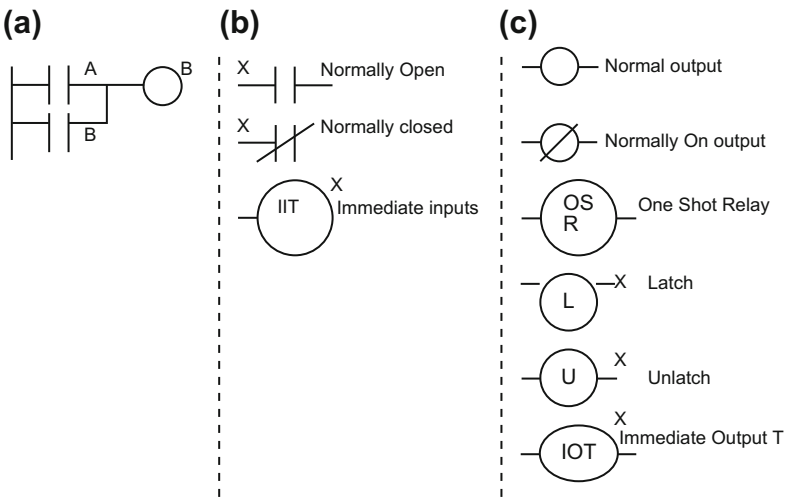


FIGURE 7.7

Relay ladder logic representations: (a) a seal-in circuit; (b) ladder logic inputs; (c) ladder logic outputs.

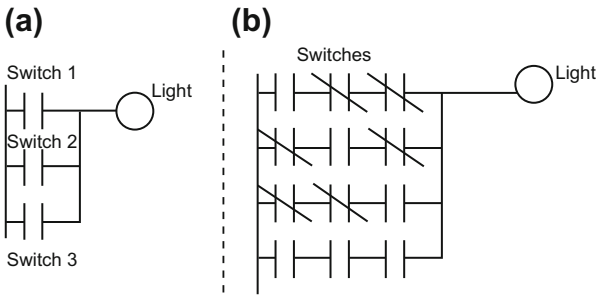


FIGURE 7.8

A case study: (a) solution one; (b) solution two.

deenergized. The output can only be turned off using a U output. The last instruction is the IOT (Immediate Output Terminal) that will allow outputs to be updated without having to wait for the ladder logic scan to be completed. When power is applied (ON) the output X is activated for the left output, but turned off for the output on the right. An input transition on will cause the output X to go on for one scan (this is also known as a one-shot relay). When the L is energized, this X will be toggled on, and will stay on until the U coil is energized. This is like a flip-flop and stays set even when the PLC is turned off. In some PLCs, all immediate outputs do not wait for the program scan to end before setting an output.

For example, to develop (without looking at the solution) a relay-based controller that will allow three switches in a room to control a single light, there are two possible approaches. The first assumes that if any switch is on will be on the light, but all three switches must be off for the light to be off. Figure 7.8(a) displays the ladder logic solution for this. The second solution assumes that each switch can turn the light on or off, regardless of the states of the other switches. This method is more complex and involves thinking through all of the possible combinations of switch positions. This problem can be recognized as an Exclusive OR problem from Figure 7.8(b).

(2) Programming

An example of ladder logic can be seen in Figure 7.9. To interpret this diagram, imagine that the power is on the vertical line on the left-hand side; we call this the hot rail. On the right-hand side is the neutral rail. In this figure there are two rungs, and on each rung there are combinations of inputs (two vertical lines) and outputs (circles). If the inputs are opened or closed in the right combination, then the power can flow from the hot rail, through the inputs, to power the outputs, and finally to the neutral rail. An input can come from a sensor or a switch. An output will be some device outside the PLC that is switched ON or OFF, such as lights or motors. In the top rung the contacts are normally open and normally closed, which means if input A is ON and input B is OFF, then power will flow through the output and activate it. Any other combination of input values will result in the output X being off.

The second rung of Figure 7.9 is more complex; several combinations of inputs will result in the output Y turning on. On the left-most part of the rung, power could flow through the top if C is OFF and D is ON. Power could also (and simultaneously) flow through the bottom if both E and F are true. This

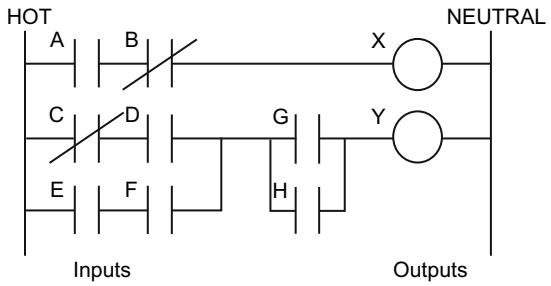


FIGURE 7.9

A simple ladder logic diagram.

would get power half way across the rung, and then if G or H is true the power will be delivered to output Y.

There are other methods for programming PLCs. One of the earliest techniques involved mnemonic instructions. These instructions can be derived directly from ladder-logic diagrams and entered into the PLC through a simple programming terminal. An example of mnemonics is shown in Figure 7.10. In this example, the instructions are read one line at a time from top to bottom. The first line 00000 has the instruction LDN (input load and not) for input 00001. This will examine the input to the PLC, and if it is OFF it will remember a 1 (or true); if it is ON it will remember a 0 (or false). The

00000	LDN	00001
00001	LD	00002
00002	AND	
00003	LD	00003
00004	LD	00004
00005	AND	
00006	OR	
00007	ST	00107
00008	END	

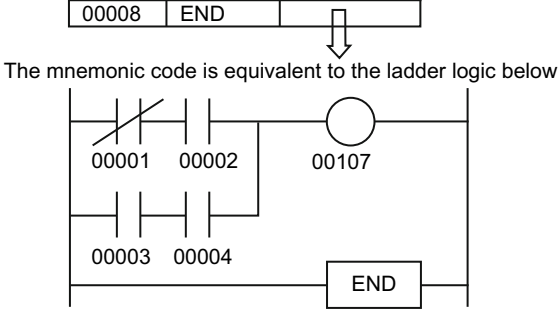


FIGURE 7.10

A mnemonic program and equivalent ladder logic.

next line uses an LD (input load) statement to look at the input. If the input is OFF it remembers a 0; if the input is ON it remembers a 1 (note: this is the reverse of the LD). The AND statement recalls the last two numbers remembered and if they are both true the result is a 1; otherwise the result is a 0. This result now replaces the two numbers that were recalled, so only one number is remembered. The process is repeated for lines 00003 and 00004, but when these are done there are now three numbers remembered. The oldest number is from the AND; the newer numbers are from the two LD instructions. The AND in line 00005 combines the results from the last LD instructions leaving two numbers in memory. The OR instruction takes the two numbers now remaining, and if either one is a 1 the result is a 1; otherwise the result is a 0 giving a final single number output. The last instruction is the ST (store output) that will look at the last value stored and if it is 1, the output will be turned on; if it is 0 the output will be turned off.

The ladder logic program in Figure 7.10 is equivalent to the mnemonic program. Ladder logic will be converted to mnemonic form before being used by the PLC. Mnemonic programming was common in the past, but now it is rarely seen.

Sequential function charts have been developed to accommodate the programming of more advanced systems. These are similar to flowcharts, but much more powerful. The example seen in Figure 7.11 is doing two different things. To read the chart, start at the top where it says Start. Below this there is the double horizontal line that says follow both paths. As a result, the PLC will start to follow the branch on the left- and right-hand sides separately and simultaneously. On the left there are two functions; firstly a power-up function which will run until it decides it is done, and the power-down function will come after. On the right-hand side is the flash function; this will run until it is done. These functions look unexplained, but each function, such as power-up, will be a small ladder logic program. This method is very different from flowcharts because it does not have to follow a single path through the flowchart.

(3) Ladder diagram instructions

Ladder logic input contacts and output coils allow simple logical decisions. Instructions extend this to allow other types of control. Most of the instructions will use PLC memory locations to get and store values, and track instruction status. Most instructions will normally become active when the input is

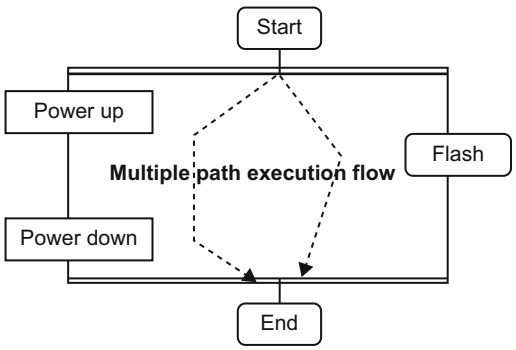


FIGURE 7.11

A sequential function chart.

true. Some instructions, such as TOF timers, can remain active when the input is off. Other instructions will only operate when the input goes from false to true; this is known as positive edge-triggered. Consider a counter that only counts when the input goes from false to true; the length of time that the input stays true does not change the instruction behavior. A negative edge-triggered instruction would be triggered when the input goes from true to false. These are rarer; unless stated, assume instructions are not edge-triggered.

Instructions may be divided into several basic groups according to their operation. Each of these instruction groups is briefly introduced in Table 7.2.

7.1.4 Basic types and important data

Programmable logic controller I/O channel specifications include total number of points, number of inputs and outputs, ability to expand, and maximum number of channels. Number of points is the sum of the inputs and the outputs. A PLC may have any possible combination of these values. Expandable units may be stacked or linked together to increase total control capacity. Maximum number of channels refers to the maximum total number of input and output channels in an expanded system. PLC system specifications to be considered include scan time, number of instructions, data memory, and program memory. Scan time is the time required by the PLC to check the states of its inputs and outputs. Instructions are standard operations (such as mathematical functions) available to PLC software. Data memory is the capacity for data storage. Program memory is the capacity for control software storage. Available inputs for programmable logic controllers include DC, AC, analog, thermocouple, RTD, frequency or pulse, transistor, and interrupt inputs. Outputs for PLC include DC, AC, relay, analog, frequency or pulse, transistor, and triac. Programming options for PLC include front panel, hand held, and computer.

Programmable logic controllers can also be specified with a number of computer interface options, network specifications, and features. PLC power options, mounting options, and environmental operating conditions also important need to be considered.

PLCs are usually one of three general types:

(1) Embedded

The embedded controllers expand their field bus terminals and transform them into a modular PLC. All support the same communication standard such as Ethernet TCP/IP. PCs and compact operating units belonging to the PLC product spectrum are also identical for all controllers.

(2) PC-based

This type of PLC is a slide-in card for a PC that extends every PC or IPC and transforms it into a fully fledged PLC. In the PC, the slide-in card needs only one PCI bus slot and runs fully independently of the operating system. PC system crashes leave the machine control completely cold.

(3) Compact

The compact PLC controller unites the functions of an operating unit and a PLC. To some extent, the compact controller already features integrated digital and analog inputs and outputs. Further field bus terminals in compact PLCs can be connected via an electrically isolated interface such as CANopen.

Table 7.2 Ladder Diagram Instructions

Group	Instruction	Function
Sequence input instructions	LOAD	Connects an NO condition to the left bus bar.
	LOAD NOT	Connects an NC condition to the left bus bar.
	AND	Connects an NO condition in series with the previous condition
	AND NOT	Connects an NC condition in series with the previous condition
	OR	Connects an NO condition in parallel with the previous condition.
	OR NOT	Connects an NC condition in parallel with the previous condition.
	AND LOAD	Connects two instruction blocks in series.
	OR LOAD	Connects two instruction blocks in parallel.
Sequence output instructions	OUTPUT	Outputs the result of logic to a bit.
	OUT NOT	Reverses and outputs the result of logic to a bit.
	SET	Force sets (ON) a bit.
	RESET	Force resets (OFF) a bit.
	KEEP	Maintains the status of the designated bit.
	DIFFERENTIATE UP	Turns ON a bit for one cycle when the execution condition goes from OFF to ON.
	DIFFERENTIATE DOWN	Turns ON a bit for one cycle when the execution condition goes from ON to OFF.
Sequence control instructions	NO OPERATION	—
	END	Required at the end of the program.
	INTERLOCK	It the execution condition for IL(02) is OFF, all outputs are turned OFF and all timer PVs reset between IL(02) and the next ILC(03).
	INTERLOCK CLEAR	ILC(03) indicates the end of an interlock (beginning at IL(02)).
	JUMP	If the execution condition for JMP(04) is ON, all instructions between JMP(04) and JME(05) are treated as NOP(OO).
	JUMP END	JME(05) indicates the end of a jump (beginning at JMP(04)).
Timer/counter instructions	TIMER	An ON-delay (decrementing) timer.
	COUNTER	A decrementing counter.
	REVERSIBLE COUNTER	Increases or decreases PV by one.
	HIGH-SPEED TIMER	A high-speed, ON-delay (decrementing) timer.
Data comparison instructions	COMPARE	Compares two four-digit hexadecimal values.
	DOUBLE COMPARE	Compares two eight-digit hexadecimal values.
	BLOCK COMPARE	Judges whether the value of a word is within 16 ranges (defined by lower and upper limits).
	TABLE COMPARE	Compares the value of a word to 16 consecutive words.

(*Continued*)

Table 7.2 Ladder Diagram Instructions *Continued*

Group	Instruction	Function
Data movement instructions	MOVE	Copies a constant or the content of a word to a word.
	MOVE NOT	Copies the complement of a constant or the content of a word to a word.
	BLOCK TRANSFER	Copies the content of a block of up to 1,000 consecutive words to a block of consecutive words.
	BLOCK SET	Copies the content of a word to a block of consecutive words.
	DATA EXCHAGE	Exchanges the content of two words.
	SINGLE WORD DISTRIBUTE	Copies the content of a word to a word (whose address is determined by adding an offset to a word address).
	DATA COLLECT	Copies the content of a word (whose address is determined by adding an offset to a word address) to a word.
	MOVE BIT	Copies the specified bit from one word to the specified bit of a word.
	MOVE DIGIT	Copies the specified digits (4-bit units) from a word to the specified digits of a word.
Shift instructions	SHIFT REGISTER	Copies the specified bit (0 or 1) into the rightmost bit of a shift register and shifts the other bits one bit to the left.
	WORD SHIFT	Creates a multiple-word shift register that shifts data to the left in one-word units.
	ASYNCHRONOUS SHIFT REGISTER	Creates a shift register that exchanges the contents of adjacent words when one is zero and the other is not.
	ARITHMETIC SHIFT LEFT	Shifts a 0 into bit 00 of the specified word and shifts the other bits one bit to the left.
	ARITHMETIC SHIFT RIGHT	Shifts a 0 into bit 15 of the specified word and shifts the other bits one bit to the right.
	ROTATE LEFT	Moves the content of CY into bit 00 of the specified word, shifts the other bits one bit to the left, and moves bit 15 to CY.
	ROTATE RIGHT	Moves the content of CY into bit 15 of the specified word, shifts the other bits one bit to the left, and moves bit 00 to CY.
	ONE DIGIT SHIFT LEFT	Shifts a 0 into the rightmost digit (4-bit unit) of the shift register and shifts the other digits one digit to the left.
	ONE DIGIT SHIFT RIGHT	Shifts a 0 into the rightmost digit (4-bit unit) of the shift register and shifts the other digits one digit to the right.
	REVERSIBLE SHIFT REGISTER	Creates a single or multiple-word shift register that can shift data to the left or right.
Increment/ decrement instructions	INCREMENT	Increments the BCD content of the specified word by 1.
	DECREMENT	Decrements the BCD content of the specified word by 1.

Table 7.2 Ladder Diagram Instructions *Continued*

Group	Instruction	Function
BCD/binary calculation instructions	BCD ADD	Adds the content of a word (or a constant).
	BCD SUBTRACT	Subtracts the contents of a word (or constant) and CY from the content of a word (or constant).
	BDC MULTIPLY	Multiplies the content of two words (or contents).
	BCD DIVIDE	Divides the contents of a word (or constant) by the content of a word (or constant).
	BINARY ADD	Adds the contents of two words (or constants) and CY.
	BINARY SUBTRACT	Subtracts the content of a word (or constant) and CY from the content of the word (or constant).
	BINARY MULTIPLY	Multiplies the contents of two words (or constants).
	BINARY DIVIDE	Divides the content of a word (or constant) by the content of a word and obtains the result and remainder.
	DOUBLE BCD ADD	Add the 8-digit BCD contents of two pairs of words (or constants) and CY.
	DOUBLE BCD SUBTRACT	Subtracts the 8-digit BCD contents of a pair of words (or constants) and CY from the 8-digit BCD contents of a pair of words (or constants)
	DOUBLE BCD MULITPLY	Multiplies the 8-digit BCD contents of two pairs of words (or constants).
	DOUBLE BCD DIVIDE	Divides the 8-digit BCD contents of a pair of words (or constants) by the 8–digits BCD contents of a pair of words (or constants)
Data conversion instructions	BCD TO BINARY	Converts 4-digit BCD data to 4-digit binary data.
	BINARY TO BCD	Converts 4-digit binary data to 4 digit BCD data.
	4 to 16 DECODER	Takes the hexadecimal value of the specified digit(s) in a word and turn ON the corresponding bit in a word(s).
	16 to 4 DECODER	Identifies the highest ON bit in the specified word(s) and moves the hexadecimal value(s) corresponding to its location to the specified digit(s) in a word.
	ASCII CODE CONVERT	Converts the designated digit(s) of a word into the equivalent 8-bit ASCII code.
Logic instructions	COMPLEMENT	Turns OFF all ON bits and turns ON all OFF bits in the specified word
	LOGICAL AND	Logically ANDs the corresponding bits of two word (or constants)
	LOGICAL OR	Logically ORs the corresponding bits of two word (or constants)
	EXCLUSIVE OR	Exclusively ORs the corresponding bits of two words (or constants)
	EXCLUSIVE NOR	Exclusively NORs the corresponding bits of two words (or constants).
Special calculation	BIT COUNTER	Counts the total number of bits that are ON in the specified block

(*Continued*)

Table 7.2 Ladder Diagram Instructions *Continued*

Group	Instruction	Function
Subroutine instructions	SUBROUTINE ENTER	Executes a subroutine in the main program.
	SUBROUTINE ENTRY	Marks the beginning of a subroutine program.
	SUBROUTINE RETURN	Marks the end of a subroutine program.
	MACRO	Calls and executes the specified subroutine, substituting the specified input and output words for the input and output words in the subroutine.
Interrupt control instructions	INTERVAL TIMER	Controls interval timers used to perform scheduled interrupts.
	INTERRUPT CONTROL	Performs interrupts control, such as masking and unmasking the interrupt bits for I/O interrupts.
Step instructions	STEP DEFINE	Defines the start of a new step and resets the previous step when used with a control bit. Defines the end of step execution when used without a control bit.
	STEP START	Starts the execution of the step when used with a control bit.
Peripheral device control instructions	BCD TO BINARY	Converts 4-digit BCD data to 4-digit binary data.
	BINARY TO BCD	Converts 4-digit binary data to 4-digit BCD data.
	4 to 16 DECODER	Takes the hexadecimal value of the specified digit(s) in a word and turn ON the corresponding bit in a word(s).
	16 to 4 DECODER	Identifies the highest ON bit in the specified word(s) and moves the hexadecimal value(s) corresponding to its location to the specified digit(s) in a word.
	ASCII CODE CONVERT	Converts the designated digit(s) of a word into the equivalent 8-bit ASCII code.
I/O unit instructions	7-SEGMENT DECODER	Converts the designated digit(s)of a word into an 8-bit, 7-segment display code.
	I/O REFRESH	Refreshes the specified I/O word.
Display instructions	MEASSAGE	Reads up to 8 words of ASCII code (16 characters) from memory and displays the message on the programming console or other peripheral device.
High-speed counter control instructions	MODE CONTROL	Starts and stops counter operation, compares and changes counter PVs, and stops pulse output.
	PV READ	Reads counter PVs and status data.
	COMPARE TABLE LOAD	Compares counter PVs and generates a direct table or starts operation.
Damage diagnosis instructions	FAILURE ALARM	Generates a non-fatal error when executed. The Error/Alarm indicator flashes and the CPU continues operating.
	SEVERE FAILURE ALARM	Generates a fatal error when executed. The Error/Alarm indicator lights and the CPU stops operating.
Special system instructions	SET CARRY	Sets Carry Flag 25504 to 1.
	CLEAR CARRY	Sets Carry Flag 25504 to 0.

7.2 CNC (COMPUTER NUMERICAL CONTROL) CONTROLLERS

CNC stands for computer numerical control, and refers specifically to the computer control of machine tools for repeatedly manufacturing complex parts. Many types of tools can have a CNC variant; lathes, milling machines, drills, grinding wheels, and so on. In an industrial production environment, all of these machines may be combined into one station to allow the continuous creation of a part.

CNC controllers are devices that control machines and processes. They range in capability from simple point-to-point linear control algorithms to highly complex nonlinear control algorithms that involve multiple axes and nonlinear movements. CNC controllers can be used to control various types of machine shop equipment, such as horizontal mills, vertical mills, lathes, turning centers, grinders, electro-discharge machines (EDM), welding machines, and inspection machines. The number of axes controlled by CNC controllers can range anywhere from one to five, with some CNC controllers configured to control more than six axes. Mounting types for CNC controllers include board, stand-alone, desktop, pendant, pedestal, and rack mount. Some units have integral displays, touch-screen displays, and keypads for controlling and programming.

The first benefit offered by all forms of CNC machine tools is improved automation. Operator intervention related to producing workpieces can be reduced or eliminated. Many CNC machines can run unattended during their entire machining cycle, freeing the operator to do other tasks. This results in reduced operator fatigue, fewer mistakes caused by human error, and consistent and predictable machining time for each workpiece. Since the machine will be running under program control, the skill level required by the CNC operator (related to basic machining practice) is also reduced compared with a machinist producing workpieces with conventional machine tools. The second major benefit of CNC technology is consistent and accurate workpieces. Today's CNC machines boast almost unbelievable accuracy and repeatability specifications. This means that once a program is verified, two, ten, or one thousand identical workpieces can be easily produced with precision and consistency. A third benefit offered by most forms of CNC machine tools is flexibility. Since these machines are run from programs, changing to a different workpiece is almost as easy as loading a different program. Once a program has been verified and executed for one production run, it can be easily recalled the next time the workpiece is to be run. This leads to yet another benefit; fast changeover. These machines are very easy to set up and run, and since programs can be easily loaded, they allow very short setup time. This is imperative with today's just-in-time production requirements.

7.2.1 CNC components and architectures

A computer numerical control (CNC) system consists of three basic components; CNC software, which is a program of instructions, a machine control unit, and processing equipment, also called the machine tool. The general relationship of these three components is illustrated in Figure 7.12.

(1) CNC software

Both the controller and the computer in CNC systems operate by means of software. There are three types of software programs required in either of them; operating system software, machine interface software, and application software.

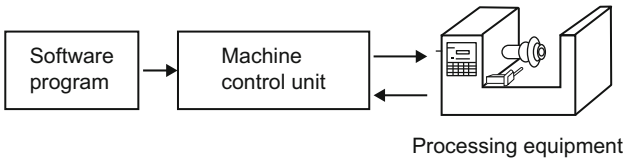


FIGURE 7.12

Elementary components of a CNC system.

The principal function of the operating system is to generate and handle the control signals which drive the machine tool axes. The machine tool interface software is used to operate the communication link between the central processing unit (CPU) and the machine tool axes to accomplish the control functions. Finally, in the application software, the program of instructions is the detailed step-by-step commands that direct the actions of the processing equipment. In machine tool applications, this program of instructions is called part programming. The individual command within the program refer to positions of a cutting tool relative to the worktable on which the work part is fixtured. Additional instructions are usually included, such as spindle speed, feed rate, cutting tool selection, and other. The program is coded in a suitable medium for submission to the machine control unit, called a controller.

(2) Machine control unit

In today's CNC technology, the machine control unit (MCU) consists of some type of computer with related control hardware that stores and sequentially executes the program of instructions by converting each command into mechanical actions of the processing equipment. The general configuration of the MCU in a CNC system is illustrated in Figure 7.13. A MCU generally consists of CPU, memory, I/O interfaces, controls of machine tool axes and spindle speed and sequence control for other machine tool functions. These subsystems are interconnected by means of an internal bus, as indicated in Figure 7.13.

The CPU, memory and I/O interfaces are described in Chapter 14 of this textbook. The MCU has two hardware subsystems machine tool controls and sequence controls which are different from normal computers. These components control the position and velocity (feed rate) of each machine axis as well as the rotational speed of the machine tool spindle. The control signals generated by the

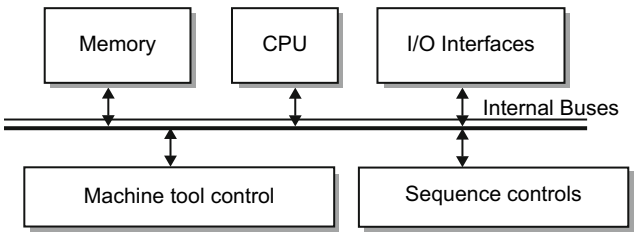


FIGURE 7.13

Subsystem blocks of MPU in a CNC system.

MCU must be converted to a form and power level suited to those position control systems used to drive each machine. The positioning system can be classified as open-loop or closed-loop, and different hardware components are required in each case. The spindle is used to drive either the workpiece, as in turning, or a rotating cutter e.g. in milling and drilling. Spindle speed is a programmed parameter for most CNC machine tools, contolled by a drive control circuit and a feedback sensor interface. In addition to control of table position, feed rate, and spindle speed, several auxiliary functions, generally ON/OFF (binary) actuations, interlocks, and discrete numerical data are also controlled by the MCU.

(3) Machine tool/processing equipment

The processing equipment transforms the starting workpiece into a complete part. Its operation is directed by the MPU, which in turn is driven by instructions contained in the part program. In most CNC systems, the processing equipment consists of a worktable and spindle, as well as the motors and controls which drive them.

(4) Auxiliary and peripheral devices

Most CNC systems also contain some auxiliary devices such as; (1) field buses, (2) servo amplifiers, or (3) power supply devices. Peripherals may include: (1) keyboards, (2) graphic display interfaces such as monitors, (3) printers, and (4) disk drivers and tape readers. The microprocessor selected is bus oriented, and the peripherals can be connected to the bus via interface modules.

Computers, especially PCs are increasingly used in factories alongside typical CNC systems. Two configurations of computers and CNC controllers are:

1. Using a PC as a separate front-end interface for displaying the control process to operators, or for entering and encoding software programs into the controller. In this case, both the PC and the controller are interconnected by I/O interface modules; usually RS-232, RS-422, RS-485 or other RS-type interfaces.
2. The PC contains the motion control chips (or board) and the other hardware required to operate the machine tool. In this case, the CNC control chip fits into a standard slot of the PC, and the selected PC will require additional interface cards and programming.

In either configuration, the advantages of using a PC for CNC machining are its flexibility in executing a variety of software at the same time as controlling the machine tool. This may be programs for shop-floor control, statistical process control, solid modeling, cutting tool management, or other computer-aided manufacturing software. They are easy to use when compared with conventional CNC and can be networked. Possible disadvantages include losing time in retrofiting the PC for CNC, particularly when installing the controls inside the PC, and and also the PC may be a less efficient means of control. Advances in the technology of PC-based CNC are likely to reduce these disadvantages over time.

7.2.2 CNC control mechanism

CNC is the process of "feeding" a set of sequenced instructions into a specially designed, programmable controller and then using it to direct the movements of a machine tool such as a milling machine, lathe, or flame cutter. The program directs the cutter to follow a predetermined path at a specific

spindle speed and feed rate, so resulting in the production of the desired geometric shape in a workpiece.

CNC controllers can operate in a number of modes, including polar coordinate command, cutter compensation, linear and circular interpolation, stored pitch error, helical interpolation, canned cycles, rigid tapping, and auto scaling. Polar coordinate command is a numerical control system in which all the coordinates are referred to a certain pole. Position is defined by the polar radius and polar angle with respect to this pole. Cutter compensation is the distance you want the CNC control to offset for the tool radius away from the programmed path. Linear and circular interpolation is the programmed path of the machine, which appears to be straight or curved, but is actually a series of very small steps along that path. Machine precision can be improved remarkably by stored pitch error compensation, which corrects for lead screw pitch error and other mechanical positioning errors. Helical interpolation is a technique used to make large-diameter holes in workpieces. It allows for high metal removal rates with a minimum of tool wear. There are machine routines such as drilling, deep drilling, reaming, tapping, boring, and so forth that involve a series of machine operations, but are specified by a single G-code with appropriate parameters. Rigid tapping is a CNC tapping feature where the tap is fed into the workpiece at the precise rate needed for a perfectly tapped hole. It also needs to retract at the same precise rate; otherwise it will shave the hole and create an out-of-specified tapped hole. Auto scaling translates the parameters of the CNC program to fit the workpiece.

Many other kinds of manufacturing equipment and manufacturing processes are controlled by other types of programmable CNC controllers. For example, a heat-treating furnace can be equipped with a controller that will monitor temperature and the furnace's atmospheric oxygen, nitrogen, and carbon and make automatic changes to maintain these parameters within very narrow limits.

(1) CNC coordinate system

To program the CNC processing equipment, a standard axis system must be defined by which the position of the workhead relative to the workpart can be specified. There are two axis systems used in CNC, one for flat and prismatic workparts and the other for parts with rotational symmetry. Both axis systems are based on the Cartesian coordinate system.

The axis system for flat and prismatic parts consists of three linear axes (x, y, z) in the Cartesian coordinate system, plus three rotational axes (a, b, c), as shown in Figure 7.14. In most machine tool

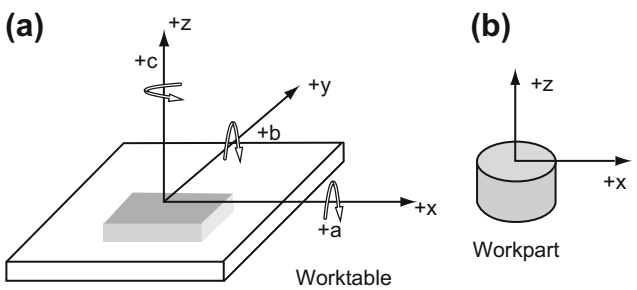


FIGURE 7.14

Coordinate systems used in numerical control: (a) for flat and prismatic work and (b) for rotational work.

applications, the x- and y-axes are used to move and position the worktable to which the part is attached, and the z-axis is used to control the vertical position of the cutting tool. Such a positioning scheme is adequate for simple numerical control applications such as drilling and punching of flat sheet metal. Programming of these machine tools consists of little more than specifying a sequence of x y coordinates. The a-, b- , and c-rotational axes specify angular positions about the x-, y- and z-axes, respectively. To distinguish positive from negative angles, the right-hand rule is used. The rotational axes can be used for one or both of the following: (1) orientation of the workpart to present different surfaces for machining or (2) orientation of the tool or workhead at some angle relative to the part. These additional axes permit machining of complex workpart geometries. Machine tools with rotational axis capability generally have either four or five axes; the three linear axes, plus one or two rotational axes. Most CNC systems do not require all six axes.

The coordinate axes for a rotational numerical control system are illustrated in Figure 7.14(b). These systems are associated with numerical control lathes and turning centers. Although the work rotates, this is not one of the controlled axes on most of these turning machines. Consequently, the y-axis is not used. The path of a cutting tool relative to the rotating workpiece is defined in the x z plane, where the x-axis is the radial location of the tool, and the z-axis is parallel to the axis of rotation of the part.

The part programmer must decide where the origin of the coordinate axis system should be located, which is usually based on programming convenience. After this origin is located, the zero position is communicated to the machine tool operator for the cutting tool to be moved under manual control to some target point on the worktable, where the tool can be easily and accurately positioned.

(2) Motion control the heart of CNC

The most basic function of any CNC controller is automatic, precise, and consistent motion control. All forms of CNC equipment have two or more directions of motion, called axes. These axes determine precisely and automatically the position of the workpiece. The two most common axis types are as mentioned before, linear (driven along a straight path) and rotary (driven along a path that the workpiece rotates around).

Instead of causing motion by manually turning cranks and handwheels as is required on conventional machine tools, CNC machines allow motion to be actuated by servomotors under control of the CNC, and guided by the part program. All the parameters of the motion; the type (rapid, linear, and circular), the axes to move, the amount, and the rate (feed rate) are programmable with almost all CNC machine tools.

Figure 7.15 shows the makeup of a linear axis of a CNC controller. In this case, a command tells the drive motor to rotate a precise number of times. The rotation of the drive motor in turn rotates the ball screw, which drives the linear axis. A feedback device at the opposite end of the ball screw allows the control to confirm that the commanded number of rotations has taken place.

Although a rather crude analogy, the same basic linear motion can be found on a common table vise. By rotating the vise crank, a lead-screw is rotated, which, in turn, drives the movable jaw on the vise. In comparison, a linear axis on a CNC machine tool is extremely precise. The number of revolutions of the axis drive motor precisely controls the amount of linear motion along the axis.

All discussions to this point have assumed that the absolute mode of programming is used, where the end points for all motions are specified from the program zero point. End points for axis motion can

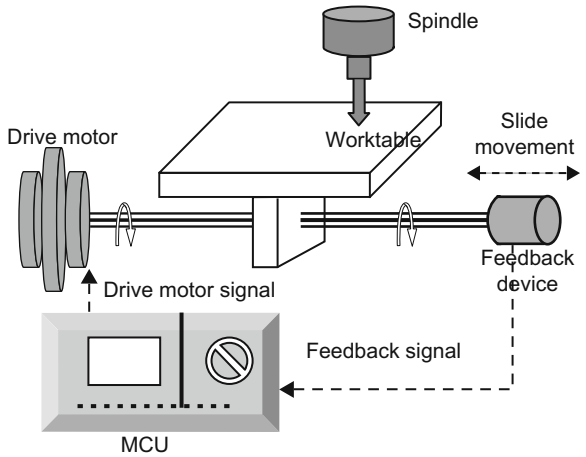


FIGURE 7.15

A CNC machine takes the commanded position from the CNC program. The drive motor is rotated a corresponding amount, which in turn drives the ball screw, causing linear motion of the axis. A feedback device confirms that the proper number of ball screw revolutions has occurred.

also be specified by an incremental mode, in which end points for motions are specified from the tool's current position, not from program zero. Although program zero point must be pinointed by one means or another, how this is done varies dramatically from one CNC controller to another. An older method is to assign program zero in the program. With this method, the programmer inputs the distance between the program zero and the starting position of the machine. A newer and better way to assign program zero is through some form of offset. Machining center control manufacturers commonly call offsets used to assign program zero fixture offsets. Turning center manufacturers commonly call offsets used to assign program zero for each tool geometry offsets.

Other motion types may exist in industrial applications, but the three most common types are available in almost all forms of CNC equipment, and are as follows:

(a) Rapid motion or positioning. This motion type is used to command motion at the machine's fastest possible rate, used to minimize non-productive time during the machining cycle. Common uses include moving the tool to and from cutting positions, moving to clear clamps and other obstructions, and in general, any non-cutting motion during the program.

(b) Straight line motion. This allows the programmer to command a perfectly straight line and the motion rate (feed rate) to be used during the movement. Straight-line motion can be used whenever a straight cutting movement is required, including when drilling, turning a straight diameter, face or taper, and when milling straight surfaces. The method by which the feed rate is programmed varies from one machine type to the next, but in general, machining centers only allow the feed rate to be specific in per-minute format (inches or millimeters per minute). Turning centers also allow feed rate to be specified in per-revolution format (inches or millimeters per revolution).

(c) Circular motion. This motion type causes the machine to make movements in the form of a circular path, and is used to generate radii during machining. All feed-rate-related points for straight-line motion apply also to the circular motion.

(3) Interpolation

For perfectly straight movement, the x- and y-axis movements must themselves be perfectly synchronized, as given in Figure 7.16. Also, if machining is to occur during the motion, a motion rate (feed rate) must also be specified. This requires linear interpolation, where the control will precisely and automatically calculate a series of very tiny single axis departures, keeping the tool as close to the programmed linear path as possible. The machine tool appears to be forming a perfectly straight-line motion. Figure 7.16(a) shows what the CNC control is actually doing during linear interpolation.

In similar fashion, many applications for CNC machine tools require that the machine be able to form circular motions for instance, when forming radii on turned workpieces between faces and turns, and milling radii on contours on machining centers. This kind of motion requires circular interpolation. As with linear interpolation, the control will do its best to generate a path that is as close to that programmed as possible. Figure 7.16(b) shows what happens during circular interpolation.

Depending on the application, other interpolation types may be required on turning centers that have live tooling. For turning centers that can rotate tools (such as end mills) in the turret and have a c-axis to rotate the workpiece held in the chuck, polar coordinate interpolation can be used to mill contours around the periphery of the workpiece. This allows the programmer to "flatten out" the rotary axis, treating it as a linear axis for the purpose of making motion commands.

(4) Compensation

All types of CNC machine tools require compensation. Though applied for different reasons on different machine types, all forms of compensation are for unpredictable conditions related to tooling. In many applications, the CNC user will be faced with several situations when it will be impossible to predict exactly the result of certain tooling-related problems, hence one or another form of compensation has to be used.

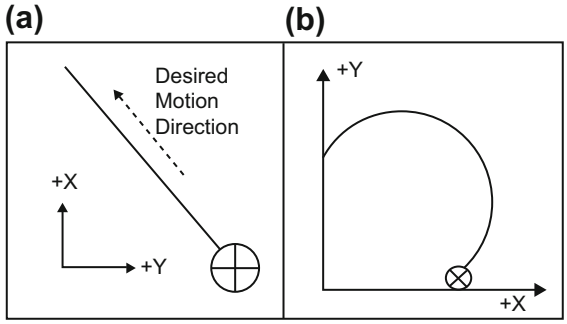


FIGURE 7.16

(a) Interpolation for actual motion generated with linear interpolation. (b) This drawing shows what happens during circular interpolation.

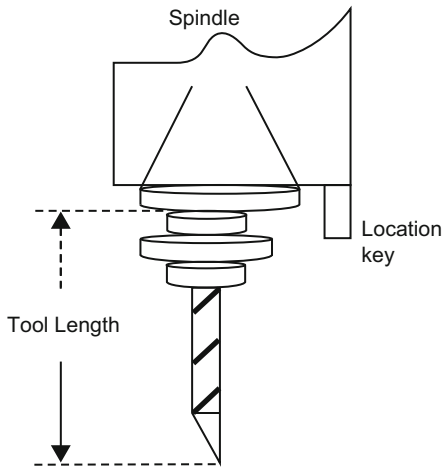


FIGURE 7.17

With tool length compensation, the tool's length compensation value is stored separately from the program. Many CNC controls allow the length of the tool to be used as the offset value.

(a) Tool length compensation

This type allows the programmer to ignore each tool length when the program is written. Instead of calculating z-axis positions based on the tool's length, the programmer simply enters tool length compensation on each tool's first z-axis approach movement to the workpiece.

During setup, the operator will input the tool length compensation value for each tool in the corresponding offset. This, of course, means the tool must first be measured. If tool length compensation is used wisely, the tool can be measured offline (in a tool length measurement gauge) to minimize setup time. Figure 7.17 shows one popular method of determining the tool length compensation value. With this method, the value is simply the length of the tool.

(b) Cutter radius compensation

Just as tool length compensation allows tool length to be ignored, so cutter radius compensation allows the programmer to forget about the cutter's radius when contours are programmed. It may be obvious that cutter radius compensation is only used for milling cutters, and only when milling on the periphery of the cutter. A programmer would never consider using cutter radius compensation for a drill, tap, reamer, or other hole-machining tool.

Without such compensation, the centerline path of all milling cutter must be entered into the program, and movements programmed according to the precise diameter of the milling cutter. This can be difficult enough with simple motions, but when contours become complicated, such calculation; become very difficult.

With cutter radius compensation, the programmer can use the coordinates of the work surface, not the tool's centerline path, so eliminating the need for many calculations. It is worth mentioning that we are now talking about manual programming. If you have a computer-aided manufacturing (CAM) system, it can probably generate a centerline path just as easily as a work surface path.

(c) Dimensional tool offsets

This compensation type applies only to turning centers. Small errors are inevitable in the positioning of a tool when it is set up. In addition, single-point turning or boring tools will start to wear during use. Both these factors will directly affect the size of the workpiece being machined.

Dimensional tool offsets (also simply called tool offsets) are required to cope with these errors, and to allow easy sizing of turned workpieces. They are installed as part of a four-digit T word. The first two digits indicate the tool station number, and the second two digits indicate the offset number to be installed. When a tool offset is installed, the control actually shifts the entire coordinate system by the amount of the offset as if the operator could actually move the tool in the turret by the amount of the offset. Each dimensional offset has two values; one for the x- and one for the z-value. The operator will have control of what the tool does in both axes as the workpiece is being machined.

(d) Other types of compensation

The compensation types shown upto this point have been for machining and turning centers, but all forms of CNC equipment also have some form of compensation to allow for unpredictable situations. For instance, CNC wire EDM (electro-discharge machine) machines have two kinds of compensation; one, called wire offset, works in a very similar way to cutter radius compensation, and keeps the wire centerline away from the work surface by the length of wire radius plus the overturn amount. It is also used to help make trim (finishing) passes using the same coordinates.

Laser cutting machines also have a similar feature to cutter radius compensation to keep the radius of the laser beam away from the surface being machined. CNC press breaks compensate for bend allowances based on the workpiece material and thickness. Generally speaking, if the CNC user is faced with any unpredictable situations during programming, it is likely that the CNC control manufacturer has come up with a form of compensation to deal with the problem.

7.2.3 CNC part programming

CNC part programming consists of designing and documenting the sequence of processing steps to be performed on a CNC machine. To do this, a CNC programmer must be able to visualize the machining operations to be performed. Then, step-by-step, commands are issued to make the machine behave accordingly. Without good visualization, the programmer will not be able to develop the movements in the program correctly. Machinists frequently make the best CNC programmers since they can easily visualize any machining operation taking place.

(1) CNC program formats

The program format is the arrangement of the data that make up the program. Commands are fed to the controller, in units called blocks or statements. A block is one complete CNC instruction, and is made up of one or more commands, such as axis or feed rate commands. The format of command information within each block is very important. Five block formats are used in CNC programming:

(a) Fixed sequential format. This requires that specific command data items be organized together in a definite order to form a complete statement or block of information. Every block must have exactly the same number of characters (words). The significance of each character depends on where it is located in the block.

Table 7.3 Common Word Prefixes Used in Word Address Format

Address Word	Function
A, B, C	Rotation about the X-, Y-, Z-axis, respectively
F	Feed rate commands
G	Preparatory commands
I, J, K	Circular interpolation X-, Y-, Z-axis offset, respectively
J	Circular interpolation Y-axis offset
K	Circular interpolation Z-axis offset
M	Miscellaneous commands
N	Sequence number
R	Arc radius
S	Spindle speed
T	Tool number
X, Y, Z	X-, Y-, Z-axis data, respectively

(b) Fixed sequential format with tab ignored. This is the same as above, except that tab codes are used to separate the characters for easier reading by humans.
(c) Tab sequential format. This is the same as the preceding format except that characters with the same value as in the preceding block can be omitted in the sequence.
(d) Word address format. This format uses a letter prefix to identify the type of word that is a single alpha character. See Table 7.3 for definition of prefix. This features an address for each data element to permit the controller to assign data to their correct register in whatever order they are received. Almost all current CNC controllers use a word address format for programming.
(e) Word address format with tab separation and variable word order. This is the same as the last format, except that characters are separated by tabs, and the characters in the block can be listed in any order.

Although the word address format allows variations in the order, the words in a block are usually given in the following order:

1. Sequence number (N-word).
2. Preparatory word (G-word; see Table 7.4 for definition of G-word).
3. Coordinates (X-, Y-, Z-words for linear axes; A-, B-, C-axes for rotational axes).
4. Feed rate (F-word).
5. Spindle speed (S-word).
6. Tool selection (T-word).
7. Miscellaneous word (M-word; see Table 7.5 for definition of M-word).
8. End-of-block (EOB symbol).

(2) Programming methodologies

Part programming can be accomplished using a variety of procedures ranging from highly manual to highly automated methods including (1) manual part programming, (2) computer-assisted part programming, and (3) conversational programming.

Table 7.4 G-Code Commands

G-Codes for Movement	
G00	It sets the controller for rapid travel mode axis motion used for point-to-point motion. Two-axis X and Y moves may occur simultaneously at the same velocity, resulting in a nonlinear dogleg cutter path. With axis priority, three-axis rapid travel moves will move the Z-axis before the X- and Y- if the Z-axis cutter motion is in the negative positive direction; otherwise the Z- axis will move last.
G01	It sets the controller for linear motion at the programmed feed rate. The controller will coordinate (interpolate) the axis motion of two-axis moves to yield a straight-line cutter path at any angle. A feed rate must be in effect. If no feed rate has been specified before entering the axis destination commands, the feed rate may default to zero inches per minute, which will require a time of infinity to complete the cut.
G02	It sets the controller for motion along an arc path at programmed feed rate in the clockwise direction. The controller coordinates the X- and Y- axes (circular interpolation) to produce an arc path.
G03	The same as G02, but the direction is counter-clockwise. G00, G01, G02, and G03 will each cancel any other of the four that might be active.
G04	It is used for dwell on some makes of CNC controllers. It acts much like the M00 miscellaneous command in that it interrupts execution of the program. Unlike the M00 command, G04 can be an indefinite dwell or it can be a timed dwell if a time span is specified.
G-Codes for Offsetting the Cutter's Center	
G40	It deactivates both G41 and G42, eliminating the offsets.
G41	It is used for cutter-offset compensation where the cutter is on the left side of the workpiece looking in the direction of motion. It permits the cutter to be offset an amount the programmer specifies to compensate for the amount a cutter is undersize or oversize.
G42	It is the same as G41 except that the cutter is on the right side looking in the direction of motion. G41 and G42 can be used to permit the size of a milling cutter to be ignored (or set for zero diameter) when writing CNC programs. Milling cut statements can then be written directly in terms of work piece geometry dimensions. Cutting tool centerline offsets required to compensate for the cutter radius can be accommodated for the entire program by including a few G41 and/or G42 statements at appropriate places in the program.
G-Codes for Setting Measurement Data Units	
G70	It sets the controller to accept inch units.
G71	It sets the controller to accept millimetre units.
G-Codes for Calling (Executing) Canned Cycles	
G78	It is used by some models of CNC controllers for a canned cycle for milling rectangular pockets. It cancels itself upon completion of the cycle.
G79	It is used by some models of N/C controllers for a canned cycle for milling circular pockets. It cancels itself upon completion of the cycle.
G80	It deactivates (cancels) any of the G80-series canned Z-axis cycles. Each of these canned cycles is modal. Once put in effect, a hole will be drilled, bored, or tapped, each time the spindle is moved to a new location. Eventually the spindle will be moved to a location where no hole is desired. Cancelling the canned cycle terminates its action.

(*Continued*)

Table 7.4 G-Code Commands *Continued*

G81	It is a canned cycle for drilling holes in a single drill stroke without pecking. Its motion is feed down (into the hole) and rapid up (out of the hole). A Z-depth must be included.
G82	It is a canned cycle for counter boring or countersinking holes. Its action is similar to G81, except that it has a timed dwell at the bottom of the Z-stroke. A Z-depth must be included.
G83	It is a canned cycle for peck drilling. Peck drilling should be used whenever the hole depth exceeds three times the drill's diameter. Its purpose is to prevent chips from packing in the drill's flutes, resulting in drill breakage. Its action is to drill in at feed rate a small distance (called the peck increment) and then retract at rapid travel. Then the drill advances at rapid travel ("rapids" in machine tool terminology) back down to its previous depth, feeds in another peck increment, and rapids back out again. Then it rapids back in, feeds in another peck increment, etc., until the final Z-depth is achieved. A total Z-depth dimension and peck increment must be included.
G84	It is a canned cycle for tapping. Its use is restricted to CNC machines that have a programmable variable-speed spindle with reversible direction of rotation. It coordinates the spindle's rotary motion to the Z-axis motion for feeding the tap into and out of the hole without binding and breaking off the tap. It can also be used with some nonprogrammable spindle machines if a tapping attachment is also used to back the tap out.
G85	It is a canned cycle for boring holes with a single-point boring tool. Its action is similar to G81, except that it feeds in and feeds out. A Z-depth must be included.
G86	It is also a canned cycle for boring holes with a single-point boring tool. Its action is similar to G81, except that it stops and waits at the bottom of the Z-stroke. Then the cutter rapids out when the operator depresses the START button. It is used to permit the operator to back off the boring tool so it does not score the bore upon withdrawal. A Z-depth must be included.
G87	It is a chip-breaker canned drill cycle, similar to the G83 canned cycle for peck drilling. Its purpose is to break long, stringy chips. Its action is to drill in at feed rate a small distance, back out a distance of 0.010 inch to break the chip, then continue drilling another peck increment, back off 0.010", drill another peck increment, etc., until the final Z-depth is achieved. A total Z-depth dimension and peck increment must be included.
G89	It is another canned cycle for boring holes with a single-point boring tool. Its action is similar to G82, except that it feeds out rather than rapids out. It is designed for boring to a shoulder. A Z-depth must be included.
G-Codes for Setting Position Frame of Reference	
G90	It sets the controller for positioning in terms of absolute coordinate location relative to the origin.
G91	It sets the controller for incremental positioning relative to the current cutting tool point location.
G92	It resets the X-, Y-, and/or Z-axis registers to any number the programmer specifies. In effect it shifts the location of the origin. It is very useful for programming bolt circle hole locations and contour profiling by simplifying trigonometric calculations.
G-Codes for Modifying Operational Characteristics	
G99	It is a no-modal-deceleration override command used on certain Bridgeport CNC mills to permit a cutting tool to move directly – without decelerating, stopping, and accelerating – from a linear or circular path in one block to a circular or linear path in the following block, provided the paths are tangent or require no sudden change of direction and the feed rates are approximately the same.

Table 7.5 M-Code Commands

Miscellaneous Commands	
M00	It is a code that interrupts the execution of the program. The CNC machine stops and stays stopped until the operator depresses the START/CONTINUE button. It provides the operator with the opportunity to clear away chips from a pocket, reposition a clamp, or check a measurement.
M01	It is a code for a conditional – or optional – program stop. It is similar to M00 but is ignored by the controller unless a control panel switch has been activated. It provides a means to stop the execution of the program at specific points in the program if conditions warrant the operator to actuate the switch.
M02	It is a code that tells the controller that the end of the program has been reached. It may also cause the tape or the memory to rewind in preparation for making the next part. Some controllers use a different code (M30) to rewind the tape.
M03	It is a code to start the spindle rotation in the clockwise (forward) direction.
M04	It is a code to start the spindle rotation in the counter-clockwise (reverse) direction.
M05	It is a code to stop the spindle rotation.
M06	It is a code to initiate the execution of an automatic or manual tool change. It accesses the tool length offset (TLO) register to offset the Z-axis counter to correspond to end of the cutting tool, regardless of its length.
M07	It turns on the coolant (spray mist).
M08	It turns on the coolant (flood).
M09	It turns off the coolant.
M10 & M11	They are used to actuate clamps.
M25	It retracts the quill on some vertical spindle N/C mills.
M30	It rewinds the tape on some N/C machines. Others use M02 to perform this function.

In manual part programming, the programmer prepares the CNC code using the low-level machine language previously described. The program is either written by hand on a form from which a punched tape or other storage medium is subsequently coded, or it is entered directly into a computer equipped with some CNC part programming software, which writes the program onto the storage medium. In either case, the manual part programming is a block-by-block listing of the machining instructions for the given job, formatted for a particular machine tool. Manual part programming can be used for both point-to-point and contouring jobs, but it is most suited to point-to-point machining operations, such as drilling. It can also be used for simple contouring jobs, such as milling and turning, when only two axes are involved. However, for complex three-dimensional machining operations, computer-assisted part programming are better.

Computer-assisted part programming systems, where the program is partially prepared automatically, allow CNC programming to be accomplished at a much higher level than manual part programming and are becoming very popular. The computer will generate the G-code level program and, once finished, the program will be transferred directly to the CNC machine tool. Whilst these systems vary dramatically from one system to the next, there are three basic steps that remain remarkably similar amongst them. First, the programmer must supply some general information.

Second, workpiece geometry must be defined and trimmed to match the workpiece shape. Third, the machining operations must be defined. The first step requires documentation information such as part name, part number, date, program file name, and possibly graphic display size for scaling purposes. The workpiece material and rough stock shape may also be required. In the second step, the programmer will describe the shape of the workpiece by using a series of geometry definition methods, probably using graphic of each geometric element as it is described and selecting from a menu of definition methods, choosing the one that makes it the easiest to define the workpiece shape. Once geometry is defined, it will need to be trimmed to match the actual shape of the workpiece to be machined. Lines that run off the screen in both directions must be trimmed to form line segments. Circles must be trimmed to form radii. In the third step, the programmer tells the computer-assisted system how the workpiece is to be machined. Usually a tool path or animation will be shown, giving the programmer a very good idea of what will happen as the program is run. This ability to visualize a program before running it is a major advantage of graphic computer-assisted systems. Finally, the programmer can command that the G-code level CNC program be created.

With conversational programming, the program is created at the CNC machine, usually using graphic and menu-driven functions. The various inputs can be checked as the program is created. When finished, most conversational controls will even show the programmer a tool path plot of what will happen during the machining cycle.

Conversational controls vary substantially from one manufacturer to the next. In most cases, they can essentially be thought of as a single-purpose computer-assisted system, and thus do provide a convenient means to generate part programs for a single machine. Some of these controls, particularly older models, can only be programmed conversationally at the machine, which means other means such as offline programming with a computer-assisted system cannot be used. However, most new models can either operate in a conversational mode or accept externally generated G-code programs. Their uses is controversial, with some companies using them exclusively and swear by their use, whilst others consider them wasteful. Everyone involved with CNC seems to have a very strong opinion (pro or con) about them. Generally speaking, conversational controls can dramatically reduce the time it takes the operator to prepare the program when compared with manual part programming.

(3) CNC part programming languages

(a) G-code commands and M-commands require some elaboration. G-code commands are called preparatory commands, and consist of two numerical digits following the “G” prefix in the word address format; see Table 7.4 for details. M-code commands are used to specify miscellaneous or auxiliary functions, as explained in Table 7.5.

(b) Automatically programmed tools (APT) are a universal computer-assisted programming system for multiple-axis contouring programming. The original CNC programming system, developed for aerospace, was first used in building and manufacturing military equipment.

APT code is one of the most widely used software tools for complex numerically controlled machining. It is a “problem oriented” language, developed for the explicit purpose of driving CNC machine tools. Machine tool instructions and geometry definitions are written in the APT language to constitute a “part program”, which is processed by the APT software to produce a cutter location file. User-supplied post processors then convert the cutter location data into a form suitable for the particular CNC machine tool. The APT system software is organized into two separate programs; the load complex and the APT processor. The load complex handles the table initiation phase and is

usually only run when changes to the APT processor capabilities are made. The APT processor consists of four components the translator, the execution complex, the subroutine library, and the cutter location editor. The translator examines each APT statement in the part program for recognizable structure and generates a new statement, or series of statements, in an intermediate language. The execution complex processes all of the definition, motion, and a related statement to generate cutter location coordinates. The subroutine library contains routines defining the algorithms required to process the sequenced list of intermediate language commands generated by the translator. The cutter location editor reprocesses the cutter location coordinates according to user-supplied commands to generate a final cutter location file.

The APT language is a statement oriented, sequence dependent language. With the exception of such programming techniques as looping and macros, statements in an APT program are executed in a strict first-to-last sequence. To provide programming capability for the broadest possible range of parts and machine tools, APT input (and output) is generalized, represented by three-dimensional geometry and tools, and is arbitrarily uniform, as represented by the moving tool concept and output data in absolute coordinates.

7.3 FLC (FUZZY LOGIC CONTROL) CONTROLLERS

The aim of this section is to introduce fuzzy logic control technologies and controllers, which include the concepts of fuzzy logic mathematics, the structures of fuzzy control systems, the methodologies of fuzzy control modeling, the types of fuzzy controllers, and the examples of fuzzy control applications, etc. This section emphasizes fuzzy control principles, methodologies, and system controllers.

7.3.1 Fuzzy control principles

An intelligent control system is one in which a physical system or a mathematical model is controlled by a combination of a knowledge-based, approximate (humanlike) reasoning, and/or a learning process structured in a hierarchical fashion. Under this simple definition, any control system which involves fuzzy logic, neural networks, expert learning schemes, genetic algorithms, genetic programming or any combination of these would be designated as intelligent control.

The term "fuzzy" refers to the ability to deal with imprecise or vague inputs. Instead of using complex mathematical equations, fuzzy logic uses linguistic descriptions to define the relationship between the input information and the output action. In engineering systems, fuzzy logic provides a convenient and user-friendly front-end for developing control programs, helping designers to concentrate on the functional objectives, not on the mathematics. Their basic concepts are listed below.

(1) Logical inference

Reasoning makes a connection between cause and effect, or a condition and a consequence. Reasoning can be expressed by a logical inference, or by the evaluation of inputs to draw a conclusion. We usually follow rules of inference that have the form: IF cause1 = A and cause2 = B THEN effect = C, where A, B, and C are linguistic variables. For example, IF "room temperature" is Medium THEN "set fan speed to Fast", where the Medium is a function defining degrees of room temperature, while Fast is a function defining degrees of speed. The intelligence lies in associating those two terms by means of

an inference expressed in heuristic IF . . . THEN terms. To convert a linguistic term into a computational framework, one needs to use the fundamentals of set theory. For the statement IF "room temperature" is Medium, we have to ask and check the question "Is the room temperature Medium?" Boolean logic, has two answers: YES or NO. Therefore, the idea of membership of an element x in a set A is a function $\mu_A(x)$ whose value indicates whether that element belongs to the set A. Boolean logic would indicate, for example, that if $\mu_A(x) = 1$, then the element belongs to set A, and if $\mu_A(x) = 0$, the element does not belong to set A.

(2) Fuzzy sets

A fuzzy set is represented by a membership function that is defined in the universe of discourse, which gives the grade, or degree, of membership within the set, of any given element. The membership function maps the elements of the universe onto numerical values between 0 and 1. A value of zero implies that the corresponding element is definitely not an element of the fuzzy set, and corresponds to the Boolean value, 0, whilst a value of unity means that the element fully belongs to the set and corresponds to the Boolean value of 1. A value of membership between zero and unity corresponds to fuzzy, or partial membership of the set. In crisp set theory, if someone is taller than 1.8 meters, we can state that such a person belongs to the "set of tall people". However, such a sharp change from the 1.7999 meters of a "short person" to the 1.8001 meters of a "tall person" not a common sense distinction.

As another example, suppose a highway has a speed limit of 65 miles/hour. Those who drive faster than 65 miles/hour belong to the set A, whose elements are violators, and their membership function has the value of 1. Those who drive slower than the limit do not belong to set A. Is sharp transition between membership and non-membership realistic? Should there be a traffic summons issued to drivers who are caught at 65.5 miles/hour? Or at 65.9 miles/hour? In practical situations there is always a natural fuzzification, where statements are analyzed, and a smooth membership curve usually better describes the degree to which an element belongs to a set.

(3) Fuzzification

Fuzzification is the process of decomposing a system input and/or output into one or more fuzzy sets. Many types of curves and tables can be used, but triangular or trapezoidal-shaped membership functions are the most common, since they are easier to represent in embedded controllers. Figure 7.18 shows a system of fuzzy sets for an input with trapezoidal and triangular membership functions. Each fuzzy set spans a region of input (or output) values graphed against membership. Any particular input is interpreted from this fuzzy set, and a degree of membership is obtained. The membership functions should overlap, in order to allow smooth mapping of the system. The process of fuzzification allows the system inputs and outputs to be expressed in linguistic terms to allow rules to be applied in a simple manner to express a complex system.

Consider a simplified implementation of an air-conditioning system with a temperature sensor. The temperature might be read by a microprocessor that has a fuzzy algorithm that processes output to continuously control the speed of a motor which keeps the room at a "good temperature"; it also can direct a vent upward or downward as necessary. Figure 7.18 illustrates the process of fuzzification of the air temperature.

There are five fuzzy sets for temperature: COLD, COOL, GOOD, WARM, and HOT.

The membership function for fuzzy sets COOL and WARM are trapezoidal, the membership function for GOOD is triangular, and those for COLD and HOT are half triangular, with shoulders

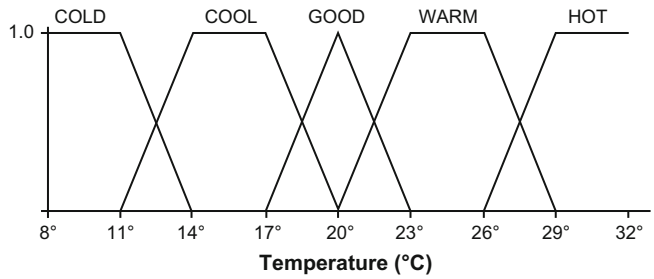


FIGURE 7.18

A temperature scale defined by fuzzy set theory.

indicating the physical limits for such a process (staying in a place with a room temperature lower than 8°C or above 32°C would be quite uncomfortable). The way to design such fuzzy sets depends solely on the designer's experience and intuition. The figure shows some non-overlapping fuzzy sets, which can indicate any nonlinearity in the modeling process. There an input temperature of 18°C would be considered COOL to a degree of 0.75 and would be considered GOOD to a degree of 0.25. To build the rules that will control the air-conditioning motor, we could watch how a human expert adjusts the settings to speed up and slow down the motor in accordance with the temperature, obtaining the rules empirically. For instance, if the room temperature is good, keep the motor speed medium; if it is warm, turn the knob of the speed to fast, and blast the speed if the room is hot. On the other hand, if the temperature is cool, slow down the speed, and stop the motor if it is cold. The beauty of fuzzy logic is the way it turns common sense, and linguistic descriptions, into a computer-controlled system. To complete this process it is necessary to understand how to use some logical operations to build the rules.

Boolean logic operations must be extended in fuzzy logic to manage the notion of partial truth truth-values between "completely true" and "completely false". A fuzziness nature of a statement such as "X is LOW" might be combined with the fuzziness statement of "Y is HIGH" and a typical logical operation could be given as X is LOW AND Y is HIGH. What is the truth-value of this AND operation? Logic operations with fuzzy sets are performed with the membership functions. Although there are various other interpretations for fuzzy logic operations, the following definitions are very convenient in embedded control applications:

$$\text{truth}(\text{ X and Y }) = \text{Min}(\text{truth}(\text{ X }), \text{truth}(\text{ Y }))$$

$$\text{truth}(\text{ X or Y }) = \text{Max}(\text{truth}(\text{ X }), \text{truth}(\text{ Y }))$$

$$\text{truth}(\text{not X }) = 1.0 - \text{truth}(\text{ X })$$

(4) Defuzzification

After fuzzy reasoning, we have a linguistic output variable that needs to be translated into a crisp value. The objective is to derive a single crisp numeric value that best represents the inferred fuzzy values of the linguistic output variable. Defuzzification is such an inverse transformation, which maps the output from the fuzzy domain back into the crisp domain. Some defuzzification methods

tend to produce an integral output, by considering all the elements of the resulting fuzzy set with their corresponding weights. Other methods take into account just the elements corresponding to the maximum points of the resulting membership functions. The following defuzzification methods are of practical importance:

(a) Centre-of-Area (C-o-A). The C-o-A method is often referred to as the Centre-of-Gravity method because it computes the centroid of the composite area representing the output fuzzy term.
(b) Centre-of-Maximum (C-o-M). In the C-o-M method only the peaks of the membership functions are used. The defuzzified, crisp, compromise value is determined by finding the place where the weights are balanced. Thus, the areas of the membership functions play no role, and only the maxima (singleton memberships) are used. The crisp output is computed as a weighted mean of the term membership maxima, weighted by the inference results.
(c) Mean-of-Maximum (M-o-M). The M-o-M is used only in cases where the C-o-M approach does not work. This occurs whenever the maxima of the membership functions are not unique and the question is which one of the equal choices one should take.

Fuzzy control strategies are derived from experience and experiment rather than from mathematical models and, therefore, linguistic implementations are accomplished much faster. They involve a large number of inputs, most of which are relevant only for some special conditions. Such inputs are activated only when the related condition prevails. In this way, little additional computational overhead is required for adding extra rules, and the resulting, rule-based structure remains understandable, leading to efficient coding and system documentation. In modern industries, are adaptive fuzzy control and fuzzy supervisory control, by applied wide in both process control and production automation.

(1) Adaptive fuzzy control

An adaptive controller in industrial applications can be defined as a controller with adjustable parameters and a mechanism for adjusting them. Despite the lack of a formal definition, an adaptive fuzzy controller has a distinct architecture, consisting of two loops; a control loop and a parameter adjustment loop (Figure 7.19).

Systems with a dominant time delay are notoriously difficult to control, and the fuzzy self-organizing controller was developed specifically to cope with dead time. To the inventors it was a further development of the original fuzzy controller. Today it may be classified as a model-reference adaptive system, an adaptive system in which the performance specifications are given by a reference model. In general the model returns the desired response to a command signal. The parameters are changed according to the model error, the deviation of the plant response from the desired response. It is still a challenge to design the adjustment mechanism such that the closed-loop system remains stable.

(2) Fuzzy supervisory control

In industrial applications, a supervisory system can be defined as a system that evaluates whether local controllers satisfy prespecified performance criteria, diagnoses causes of deviation from these criteria, plans actions, and executes the planned actions.

By process control we shall understand the automation of a large-scale industrial plant, where its complexity makes it impossible to achieve the complete satisfaction of a particular control specification. Typical goals for a supervisory controller are safe operation, highest product quality, and most

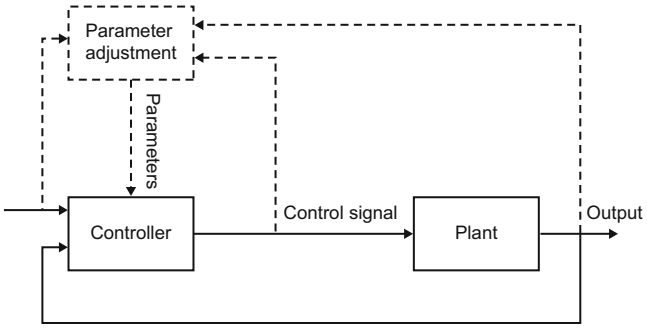


FIGURE 7.19

Adaptive control system. The inner loop (solid line) is an ordinary feedback control loop around the plant. The outer loop (dashed line) adjusts the parameters of the controller.

economic operation. All three goals are usually impossible to achieve simultaneously, so they must be prioritized; presumably, safety gets the highest priority.

7.3.2 Fuzzy control modeling

Although the development of fuzzy control strategies comes from experience and experiments rather than from mathematical models, the conventional control approach requires modeling of the physical reality. Basically, there is simply a "language difference" between fuzzy and conventional control; differential equations are the language of conventional control, and rules are the language of fuzzy control. To clearly explain this difference, a robot control system is shown in Figure 7.20. In this figure, the fuzzy control variables make it possible to design, optimize and implement tailor-made (i.e. heuristically based, nonlinear) dream controls with specific parameters and boundary conditions. Compared with conventional control concepts, the implementation of fuzzy control requires relatively little effort and is quite transparent from an engineering point of view. Utilization of fuzzy logic as a design and optimization tool has therefore proven useful in industrial control systems.

Basically, then, the role of modeling in fuzzy control design is quite similar to its role in conventional control system design. In the former there is a focus on the use of rules to represent how to control the plant rather than differential equations. From a theoretical point of view, fuzzy logics are rule-based, or use knowledge that can be used to identify both a model, as a universal approximation, as well as a nonlinear controller. The most relevant information about any system comes from either a mathematical model, sensory input/output data, or human expert knowledge. The common factor in all these three sources is knowledge.

Both fuzzy control system design and fuzzy controller design essentially amount to (1) choosing the fuzzy controller inputs and outputs, (2) choosing the preprocessing that is needed for the controller inputs and possibly postprocessing that is needed for the outputs, and (3) designing each of the four components of the fuzzy controller, as shown in Figure 7.21. Accordingly, in industrial control, the three methods below may be used for fuzzy control modeling.

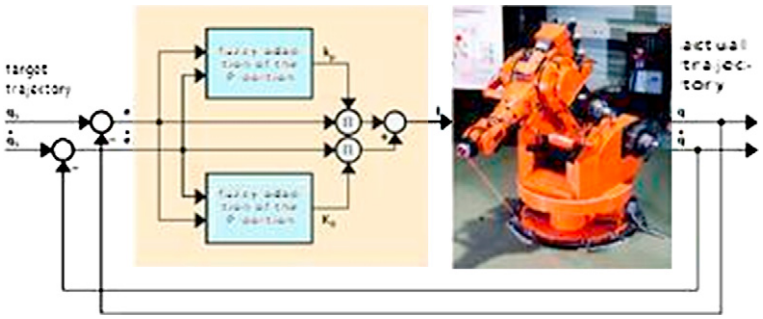


FIGURE 7.20

Structured variables of a fuzzy control system.

(1) Experimental method

By experimenting and determining how the process reacts to various inputs, one can characterize an input-output table. Graphically the method is equivalent to plotting some discrete points on an input-output curve, using the horizontal axis for input and the vertical axis for output. These are disadvantages to this approach; the process equipment may not be available for experimentation, the procedure could be very costly, and for a large number of input values it is impractical to measure the output, and interpolation between measured outputs would be required. One must also be careful to determine the expected ranges of inputs and outputs to make sure that they fall within the range of the measuring instruments available.

(2) Mathematical modeling

Control engineering requires an idealized mathematical model of the controlled process, usually in the form of differential or difference equations. Laplace transforms and z-transforms are respectively used. In order to make mathematical models simple enough, certain assumptions are made, one of which is

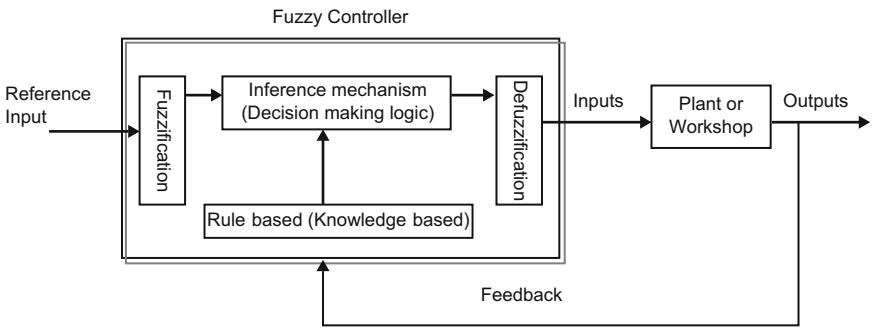


FIGURE 7.21

The general architecture of fuzzy control systems.

that the process is linear, that is, its output is proportional to the input. Linear techniques are valuable because they provide good insight, and since no general theory for the analytic solution of nonlinear differential equations exists, these are no comprehensive analysis tools for nonlinear dynamic systems. Another assumption is that the process parameters do not change over time (that is, the system is time-invariant) despite system component deterioration and environmental changes. Other problems arising when developing a meaningful and realistic mathematical description of an industrial process are; (1) poorly understood phenomena, (2) inaccurate values of various parameters, (3) model complexity.

(3) Heuristic method

The heuristic method consists of modeling and understanding in accordance with previous experience, rules-of-thumb and often-used strategies. A heuristic rule has the logical form: IF <condition> THEN <consequence>, or in a typical control situation; IF <condition> THEN <action>. Rules associate conclusions with conditions. The heuristic method is therefore similar to the experimental method of constructing a table of inputs and corresponding output values where instead of having crisp numeric values of input and output variables, one uses fuzzy values: IF input-voltage = Large THEN output-voltage = Medium. The advantages are: (1) the assumption of linearity is not required, and (2) heuristic rules can be integrated into the control strategies of human operators.

As discussed in subsection 7.3.1, there are standard choices for fuzzification and defuzzification interfaces. Designer inference mechanisms may have their preferred and may use this for many different processes. Hence, the main design work is to define the rule-base, which is constructed so that it represents a human expert "in-the-loop". The information that is loaded into the rules may come from an actual human expert who has learned how best to control the plant (or workshop) processes. In other situations there may be no such human expert, and the control engineer will simply study the plant process (perhaps using modeling and simulation) and write down a set of control rules derived from observations.

As an example, in cruise control, it is clear that anyone who has experience of driving a car can practice regulating the speed about a desired set-point and load this information into a rule-base. One rule that a human driver may use is "If the speed is lower than the set-point, then press down further on the accelerator pedal". A rule representing more detailed information about how to regulate the speed would be "If the speed is lower than the set-point AND it is rapidly approaching the set-point, then release the accelerator pedal by a small amount". This second rule characterizes our knowledge about how to make sure that we do not overshoot our desired goal (the set-point speed). Generally speaking, very detailed expertise in the rule-base results in better performance.

At this point we will examine how to use fuzzy systems for estimation and identification to study the problem of how to construct a fuzzy system from numerical data, and how to construct a fuzzy system that will serve as a parameter estimator.

To design fuzzy control systems or controllers in a plant, we need data-fitting technology that shows roughly how the input-output mapping of the estimator should behave. One way to generate this is to use a simulation test platform. A set of simulations can then be conducted, each with a different value for the parameter to be estimated. Appropriate data pairs can be gathered, that allow for the construction of a fuzzy estimator. For some plants it may be possible to perform this procedure with actual experimental data (by physically adjusting the parameter to be estimated). In a similar way, one could construct fuzzy predictors by using the basic function identification and approximation.

Some fundamental issues now need to be considered, including fitting a function to input-output data, incorporating linguistic information into the function to match it to the data measuring a function fits the data; and how to choose a data set for an engine failure estimation problem (a type of parameter estimation problem in which when estimates of the parameters take on certain values).

The main identification and estimation methods are:

1. The least squares method, which is for tuning fuzzy systems and training fuzzy systems.
2. The gradient method, which can be used to train a standard fuzzy system, especially a standard Takagi-Sugeno fuzzy system.
3. The clustering method, which contains two techniques for training fuzzy systems based on clustering.
4. Hybrid methods for training fuzzy systems can be developed by combining the above methods.

The first technique uses "c-means clustering" and least squares to train the premises and consequents, respectively, of the Takagi-Sugeno fuzzy system; while the second uses a nearest-neighborhood technique to train standard fuzzy systems. Most work in fuzzy control has to date focused only on its advantages and has not considered possible disadvantages it (the reader should therfore be cautious when reading the literature). For example, the following questions may need examining when gathering heuristic control knowledge:

1. Will the behaviors that are observed by a human expert and used to construct the fuzzy controller include all situations that can occur due to disturbances, noise, or plant parameter variations?
2. Can the human expert realistically and reliably foresee problems that could arise from closed-loop system instabilities or limit cycles?
3. Will the human expert be able to effectively incorporate stability criteria and performance objectives (e.g., rise-time, overshoot, and tracking specifications) into a rule-base to ensure that reliable operation can be obtained?

7.3.3 Fuzzy industrial controllers

This subsection briefly introduces three important types of fuzzy logic controllers in industrial control applications.

(1) Embedded fuzzy controllers

The concept of fuzzy logic makes feasible the use of a fuzzy controller built on a microcontroller or microprocessor chipset or industrial computers. Manufacturers have recognized the power of fuzzy logic and have created fuzzy kernels and support tools. In some industrial applications, such as motion control, electrical drives, temperature and humidity stabilization, the fuzzy controller receives information from the controlled process via analog-digital converters and controls it through digital-analog converters, as in Figure 7.22.

(2) Fuzzy three-term (PID-like) controllers

The industrial three-term (PID-like) controller constitutes the backbone of industrial control, having been in use for over a century. They can take a number of forms, from the early mechanical to later

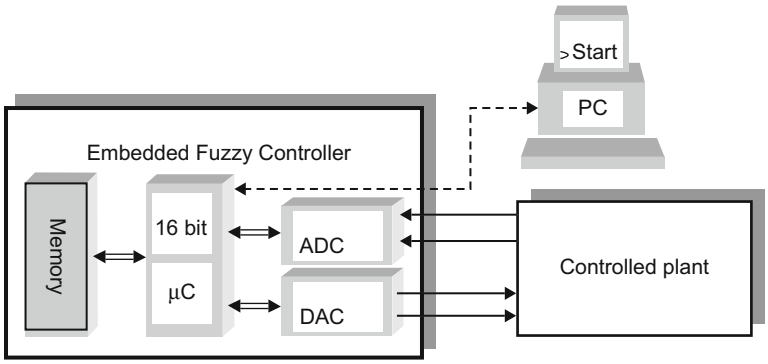


FIGURE 7.22

The concept of embedded fuzzy controller (ADC: analog-digital converter; DAC: digital-analog converter).

hydraulic, pneumatic, analog and digital versions. The modern form takes the form of multitasking discrete-time three-term (PID) algorithms embedded in almost all industrial PLCs and RTUs.

The advent of fuzzy control motivated many researchers to reconsider this controller in the hope that "fuzzification" would improve its domain of effectiveness. This is a case of a technology retrofit, in which computational intelligence is applied to improve a well-known device. The result has been a new generation of intelligent three-term (PID-like) controllers that are more robust. Fuzzy logic can be applied in a number of ways. One obvious approach is to fuzzify the gains of the three-term (PID) controller by establishing rules whereby these gains are varied in accordance with the operating state of the closed system.

It is often more convenient to partition the three-term controller into two independent fuzzy sub-controllers that separately generate the signals uPD and uI that correspond to the proportional plus derivative term and the integral term respectively. The result is a fuzzy proportional plus derivative sub-controller FPD (fuzzy proportional and differentiate) in parallel with a fuzzy integral controller FPI (fuzzy proportional and integral) as shown in Figure 7.23. Both sub-controllers are fed with the error and its derivative. The second sub-controller requires an integrator (or accumulator) to generate the integral term.

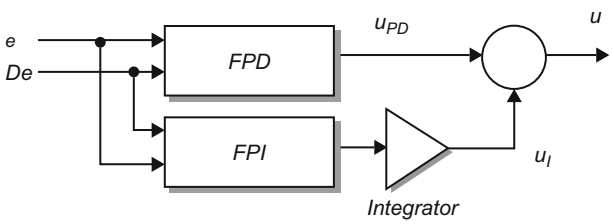


FIGURE 7.23

Decomposition of a three-term (PID-like) controller into two sub-controllers.

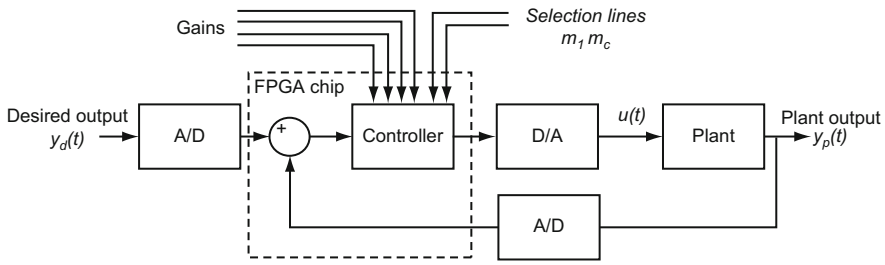


FIGURE 7.24

Layout of the FPGA-based fuzzy logic controller in a unity feedback control system.

(3) FPGA-based fuzzy logic controllers

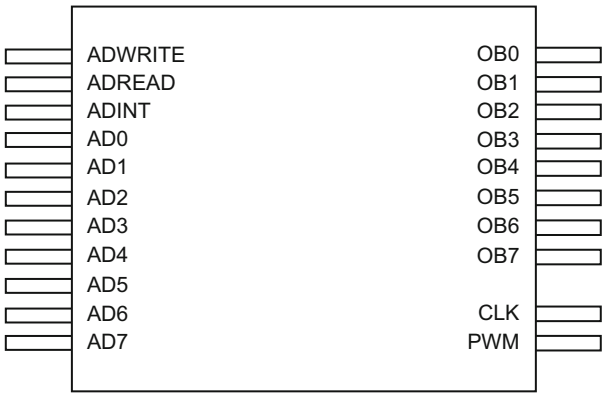
A recent hardware design and implementation of fuzzy logic controllers builds all fuzzy control modules on a field-programmable gate array (FPGA) chipset. The general layout of the controller chip in a unity feedback control system is shown in Figure 7.24. Generally, the proposed controller accepts both the output of the plant (yp) and the desired output (yd), as digital signals, and delivers a digital control action signal as an output. The design also accepts four digital signals that represent the gain coefficients needed by the controller (proportional gain Kp, derivative gain Kd, integral gain Ki, and output gain Ko), and two one-bit signals to select the type of the controller. Figure 7.25 shows a view of its FPGA chipset.

7.3.4 A case study of fuzzy industrial controls

The schematic diagram of a pilot pressure control plant is shown in Figure 7.26(a), where PT denotes pressure transmitter, PI denotes pressure indicator, "V to I" denotes voltage to current converter, "I to V" denotes current to voltage converter, and "I to P" denotes current to pressure converter. It consists of a miniature pressure tank, whose inlet is connected to an air compressor through a 50 mm control valve. At the bottom of the tank, an outlet is provided with a manually operated gate valve to allow air flow at a constant rate. An accurate pressure transmitter connected to the pressure tank is used to measure tank pressure and provide an output current in the range of 4 to 20 mA. In this closed-loop pressure-regulating system, the inlet air flow rate is manipulated by changing the control valve position in such a way as to attain the desired pressure.

An increasing sensitive-type nonlinear electro-pneumatic control valve, whose characteristics can be seen in Figure 7.26(b), is used for inlet flow manipulation. The pressure control system also has a dead time of 1.4 seconds, which has been calculated from an open-loop experiment.

The prime objective of the proposed design is to improve stability and robustness. Industrial pressure plants are usually connected to several parallel operating plants, such as in an industrial steam-generating boiler connected to a steam network of a high-pressure header, intermediate pressure level, and low pressure level. To provide a smooth and trouble-free supply for the steam-consuming processes, pressure should be stabilized at close to the set level. In addition, industrial pressure plants may be exposed to frequent load changes caused by the trips and start-ups of the steam-consuming processes. To overcome such load disturbances, and to stabilize the output over the input variations,



CLK:	Input I/O, clock signal input
OB0 7:	Input I/O, objective temperature
ADWRITE:	Output I/O, start A/D converter
ADINT:	Input I/O, notify finish of A/D converter
ADREAD:	Output I/O, get data from A/D
AD0 7:	Input I/O, A/D data periphal
PWM:	Output I/O, heater control periphal

FIGURE 7.25

A pin-out diagram of the FPGA chip of a FPGA-based fuzzy logic controller.

the controller needs an online tuning method. Hence, the supervisory-system-based online tuning method to improve the stability and robustness of the controller is emphasized in its design.

The control structure consists of a simple, upper-level, rule-based, (supervisory) controller and a lower-level, rule-based, (direct) fuzzy controller. A standard type fuzzy logic controller has been applied in both upper and lower levels of this hierarchical control structure. The supervisory fuzzy system determines the scaling factor for the direct fuzzy controller at each sampling time by evaluating the inputs e(k) and u(k).

(1) The direct fuzzy controller

The design of the direct fuzzy controller is based on research into this process and underlying domain knowledge about pressure tank systems. The universe of discourse of each input and output are divided into adjacent intervals with overlap. The membership functions are introduced to characterize each interval, and, using fuzzy logic, a continuous input and output mapping is made. The universe of discourse of inputs and output are determined from the operating range of the process. Values are finely tuned by experiment to improve controller performance. In the case of the direct fuzzy controller, five membership triangular functions with 50% overlap have been chosen for the inputs E, ¢E, and output (u). The linguistic descriptions of input membership functions are NB (negative big), NS (negative small), ZE (zero), PS (positive small) and PB (positive big). The output membership functions are VS (very small), SM (small), MD (medium), HI (high), and VH (very high).

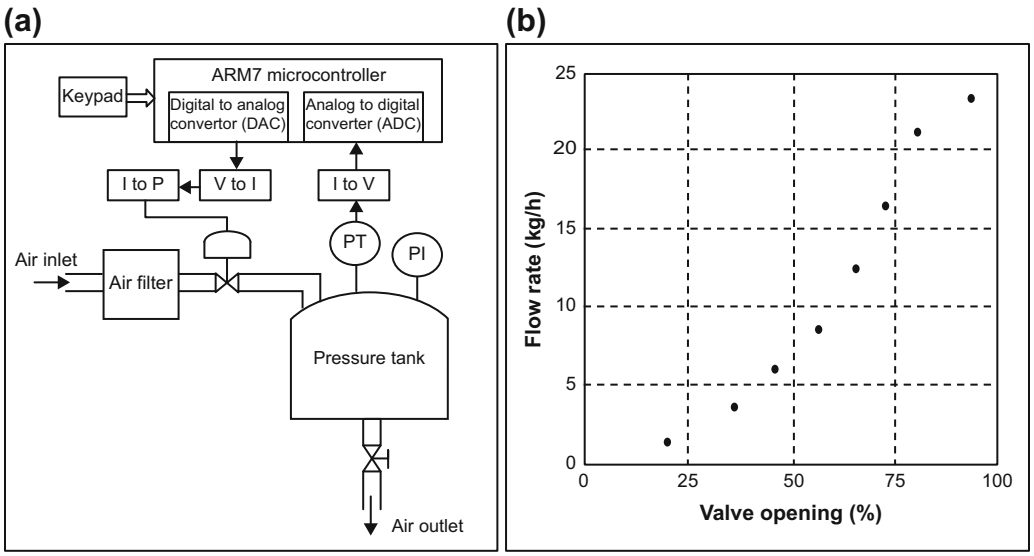


FIGURE 7.26

Experimental setup of a pressure control system. (a) Schematic diagram of the pressure control plant; (b) control valve characteristics.

(Courtesy of N. Kanagaraj et al., 2009.)

The rule-base of the direct fuzzy controller relates the premise (E and ¢E) to consequent (u). The values of E and ¢E at each sampling time are determined by:

$$E(k) = e(k)K_e(k)$$

$$¢E(k) = ¢e(k)K_d(k)$$

$$¢e = e(k\text{-}1) - e(k)$$

where E and ¢E are the error and change in error inputs of the direct fuzzy controller with scaling factor taken into account, K_e and K_d are the scaling factors, ¢e is the change in error, and k represents the sampling instant. The structure of the control rules of the direct fuzzy controller with two inputs and an output is expressed as: "If E is PS and ¢E is NS, then u is MD."

(2) Supervisory fuzzy controller

The rule-based supervisory fuzzy controller is designed to tune the input scaling factor of the direct fuzzy controller in a closed-loop system. The online scaling factor modification is adapted in the proposed control scheme, thus enhancing controller performance and significantly reducing the need for human intervention in real industrial control applications. The universe of discourse of input and output of the supervisory fuzzy controller is based on the maximum allowable range of the process. Three, triangle-shaped, membership functions are used for both the inputs and outputs. The membership function of input e is denoted by NE (negative), ZE (zero), and PE (positive); the input u and outputs K_e and K_d are denoted by LOW (low), MED (medium), and HIG (high).

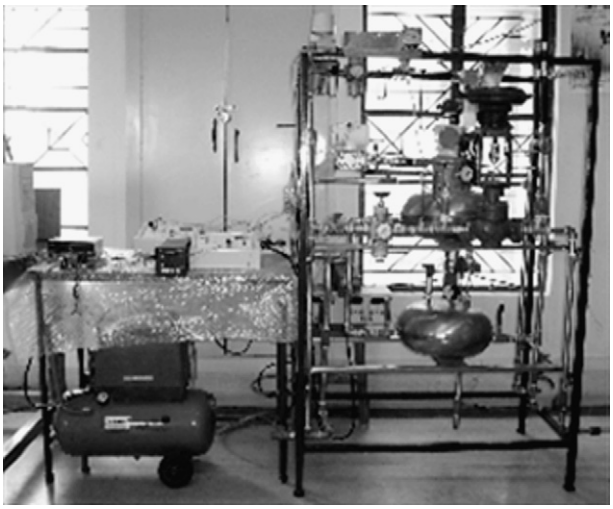


FIGURE 7.27

Experimental setup of a pressure control system: photograph of the experimental system.

(Courtesy of N. Kanagaraj et al., 2009.)

The rule-base of the supervisory fuzzy controller has been designed in the light of operative knowledge about the process. A typical fuzzy control rule of the proposed supervisory system is expressed as: "If e is NE and u is MED; then K_e is LOW and K_d is HIG".

This proposed fuzzy control scheme has been tested for a real-time pressure control application using an ARM7-based embedded microcontroller board. ARM7 is a 32-bit advanced reduced instruction set computing (RISC) architecture processor, having one megabyte (MB) on-board flash memory, network application capable processor (NACP) features, RS-232 trans-receiver and onboard analog-to-digital converter (ADC) and digital-to-analog converter (DAC) for real-time interfacing. A photograph of the ARM7 embedded microcontroller board and the experimental set-up is shown in Figure 7.27.

Problems

1. This chapter introduces scanning as one of the PLC control mechanisms. In fact, the scanning described in this chapter is the same as polling. There is another method, interrupts, that can be used to replace scanning way. Please:
   1. write the control story as if using interrupts to replace scanning in the systems given in Figures 7.3 and 7.4;
   2. for a local PLC and a central PLC (see Figure 7.1): which is better suited to scanning, and which is better suited to using interrupts?
2. With whatever programming language you are familiar, write two segments of program for the control stories given by Figure 7.3 and Figure 7.6.
3. Based on Figure 7.8, please plot a ladder logic diagram for a relay based controller that will allow three switches in a room to control a single light.
4. Please transfer the mnemonic program in Figure 7.10 into a program written in any other programming language.

5. Draw one system architecture diagram and one data flow diagram for two basic configurations between a computer (PC) and a CNC controller given below:
   1. the PC is used as a separated front end interface for displaying the control process to the operator or for entering and encoding software programs in the CNC controller;
   2. the PC contains the motion control chip (or board) and the other hardware required to operate the machine tool.
6. In terms of the G code and M code in Table 7.4 and Table 7.5, write a piece of program executed in Figure 7.15 to control the drive motor to rotate a precise number of times.
7. Find a mechanical part from, say, a bicycle or a washing machine; then think about its part programming code executed by a CNC controller.
8. By means of membership function concept in set theory, draw the membership functions for the input and output universes of discourse. Be sure to label all the axes and include both the linguistic values and the linguistic numeric values. Explain why this choice of membership functions also properly represents the linguistic values.
9. Suppose that e(t) 0 and d/dt e(t) π/8 π/32 (0.294). Which rules are on? Assume that minimum is used to represent the premise and implication. Provide a plot of the implied fuzzy sets for the two rules that result in the highest peak on their implied fuzzy sets (i.e., the two rules that are "on" the most).
10. For problem 9 above, use the three defuzzification methods given in this chapter (Centre of Area, Centre of Maximum and Mean of Maximum) for defuzzification; and find the output of the fuzzy controller. First, compute the output assuming that only the two rules found in problem 9 above are on. Next, use the implied fuzzy sets from all the rules that are on (note that more than two rules are on). Note that for computation of the area under a Gaussian curve, you will need to write a simple numerical integration routine (e.g., based on a trapezoidal approximation) since there is no closed form solution for the area under a Gaussian curve.
11. Design an adaptive fuzzy controller for the cruise control problem described in subsection 7.3.2. Use the adaptive parallel distributed compensator approach, which can be found in some of the reference books listed below. Simulate the system to show that you achieve good performance. Fully specify your algorithm and give values for all the design parameters.
12. Design a supervisory fuzzy controller for the cruise control problem described in subsection 7.3.2. Use the supervised fuzzy learning control which can be found in some of the reference books listed below. Simulate the system to show that you achieve good performance. Fully specify your algorithm and give values for all the design parameters.
13. Consider the differences between your solutions for problem 11 and problem 12; and work out the advantages and disadvantages of adaptive fuzzy control strategies and fuzzy supervisory control strategies.
14. Explain the control requirements for a pilot pressure control plant including its nonlinear electro pneumatic control valve, as shown in Figure 7.26(a) and (b). Then list all control rules with linguistic description in a table.
15. Try your best to name the instruments in Figure 7.27 as far as possible.

Further Reading

Aerotech (http://www.aerotech.com). Soft PLC. http://www.aerotech.com/ACSBrochure/plc.html. Accessed: January 2008.

AMCI (http://www.amci.com). What is PLC. http://www.amci.com/tutorials/ tutorials what is programmable logic controller.asp. Accessed: January 2008.

ARP (http://www.arptech.com.au). CNC specifications. http://www.arptech.com.au/specs/cncspecs.htm. Accessed: January 2008.

Automation Direct (http://web4.automationdirect.com). PLC hardware. http://web4.automationdirect.com/adc/Overview/Catalog/PLC Hardware . Accessed: January 2008.

AXYZ (http://www.axyz.com). CNC router specifications. http://www.axyz.com/table 4000.html. Accessed: January 2008.

BALDOR (http://www.baldor.com). Servo control facts. http://www.baldor.com/pdf/manuals/1205 394.pdf. Accessed: January 2008.

BMJ (http://www.bmjmold.com). CNC milling. http://www.bmjmold.com/cnc milling bmj.htm. Accessed: January 2008.

Brock Solutions (http://www.brocksolutions.com). S88 batch control; model and standard. http://www.brocksolutions.com/food cp/S88%20Brock.pdf. Accessed: January 2008.

CMC (http://www.cmccontrols.com). Soft PLCs. http://www.bin95.com/plc training.htm. Accessed: January 2008.

CNC (http://cnc.fme.vutbr.cz). Computer numerical control. http://cnc.fme.vutbr.cz/. Accessed: January 2008.

CNC Academy (http://www.cnc academy.com). CNC programming. http://www.cnc academy.com/cnc programming articles/cnc programming articles.htm. Accessed: January 2008.

Robert E King. Computational Intelligence in Control Engineering. Marcel Dekker. 1999.

Ian Shaw. Fuzzy Control of Industrial Systems: Theory and Applications. Kluwer Academic. 1998.

Kevin Passino, Stephen Yurkovich. Fuzzy Control. Addison Wesley Longman. 1998.

N. Kanagaraj et al. A fuzzy logic based supervisory hierarchical control scheme for realtime pressure control. International Journal of Automation and Computing, 6(1) (2009) 88 96.

Jan Jantzen. A tutorial on adaptive fuzzy control. http://www.eunite.org. Accessed: February 2009.

Ferenc Farkas, Sandor Halasz. Embedded fuzzy controller for industrial applications. Acta Polytechnica Hungarica, 3(2) (2006) 41 64.

Marcelo Godoy Simoes. Introduction to fuzzy control. http://egweb.mines.edu/faculty/msimoes/tutorials/Introduction fuzzy logic/Intro Fuzzy Logic.pdf. Accessed: February 2009.

Mohammed Hassan, Waleed Sharif. Design of FPGA based PID like fuzzy controller for industrial applications. LAENG International Journal of Computer Science, 34(2) (2007) 10 17.

Wikipedia (http://en.wikipedia.org/wiki). Fuzzy control system. http://en.wikipedia.org/wiki/Fuzzy system. Accessed: February 2009.

CHAPTER

Industrial process controllers

8

Industrial process controllers are used in chemical, mechanical, microelectronic, material, pharmaceutical, and electrochemical processes. They receive inputs from sensors, meters and so on, provide control functions, and deliver output control signals to controlled devices in plants or workshops. There are three generic types; PID controllers, batch controllers and servo controllers.

Some industrial process controllers are printed-circuit boards that can be attached to an enclosure or plugged directly into a computer backplane. Others are designed to attach to a panel or bolt onto a chassis, wall, cabinet, or enclosure. Rack-mounted industrial process controllers include hardware such as rail guides, flanges, or tabs, and may fit a standard telecommunications rack. Bench-top or floor-standing units have a full casing or cabinet and an integral interface that includes either a digital front panel or analog components such as knobs, switchers, jumpers, or meters.

8.1 PID (PROPORTIONAL-INTEGRAL-DERIVATIVE) CONTROLLERS

A PID (proportional-integral-derivative) controller is a standard feedback-loop component that measures an output of a process and controls an input, with the goal of maintaining the output at a target value, called the set-point. An example of a PID application is the control of a process temperature, although it can be used to control any measurable variable which can be affected by manipulating some other process variable, such as pressure level, flow rate, chemical composition, force, speed, or a number of other variables. Automobile cruise control is an example of a PID application area outside the process industries.

8.1.1 PID control mechanism

PID can be described as a group of rules by which the precise regulation of a closed-loop feedback control system is obtained. Closed-loop feedback control means that the real-time measurement of the process being controlled is constantly fed back to the controller, to ensure that the target value is, in fact, being realized. The mission of the controlling device is to make the measured value, usually known as the process variable, equal to the desired value, usually known as the set-point. It uses the control algorithm we know as three-mode; proportional + integration + derivative. Figure 8.1 is a schematic of a feedback control loop with a PID controller.

The most important element of the algorithm, proportional control, determines the magnitude of the difference between the set-point and the process variable, which is defined as error. Then it applies appropriate proportional changes to the controller output to eliminate error. Many control systems will in fact, work quite well with only proportional control. Integral control examines the offset of set-point and the process variable over time and corrects it when and if necessary. Derivative control monitors

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10008-7

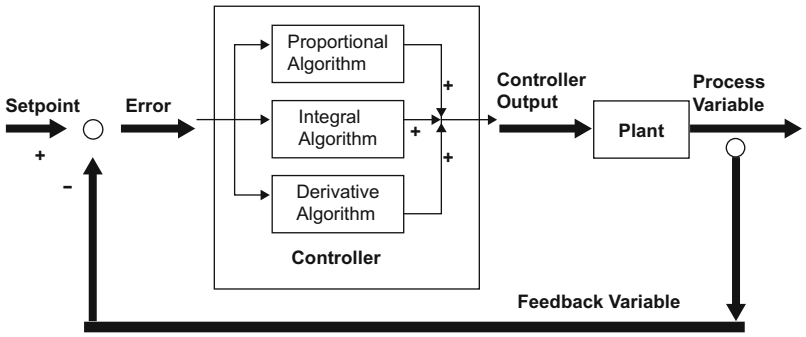


FIGURE 8.1

A schematic of a feedback control loop with a PID controller.

the rate of change of the process variable and makes changes to the controller output to accommodate unusual changes.

Each of the three control functions is governed by a user-defined parameter, which vary immensely from one control system to another, and, as such, need to be adjusted to optimize the precision of control. The process of determining the values of these parameters is known as PID tuning which can be done by a number of different methods. Certain PID tuning methods require more equipment than others, but usually result in more accurate results with less effort.

8.1.2 PID controller implementation

Like all feedback controllers, a PID controller usually uses an actuator to affect the process and a sensor to measure it. Virtually all PID controllers determine their output by observing the error between the set-point and a measurement of the process variable. Errors occur when an operator changes the set-point intentionally or when a disturbance or a load on the process changes the process variable accidentally. The mission of the controller is to eliminate the error automatically.

Consider, for example, the mechanical flow controller depicted in Figure 8.2(a). A portion of the water flowing through the tube is bled off through the nozzle on the left, driving the spherical float upwards in proportion to the flow rate. If the flow rate slows because of a disturbance such as a leakage, the float falls and the valve opens until the desired flow rate is restored.

In this example, the water flowing through the tube is a continuous process, and its flow rate is the process variable that is to be measured and controlled. The lever arm serves as the controller, taking the process variable measured by the float's position and generating an output that moves the valve's piston. Adjusting the length of the piston rod sets the desired flow rate; a longer rod corresponds to a lower set-point and vice versa.

Suppose that at time t, the valve is open by V(t) inches and the resulting flow rate is sufficient to push the float to a height of F(t) inches. This process is said to have a gain of $G_p = F(t)/V(t)$. The gain shows how much the process variable changes when the controller output changes. In this case,

$$F(t) = G_p\, V(t) \tag{1}$$

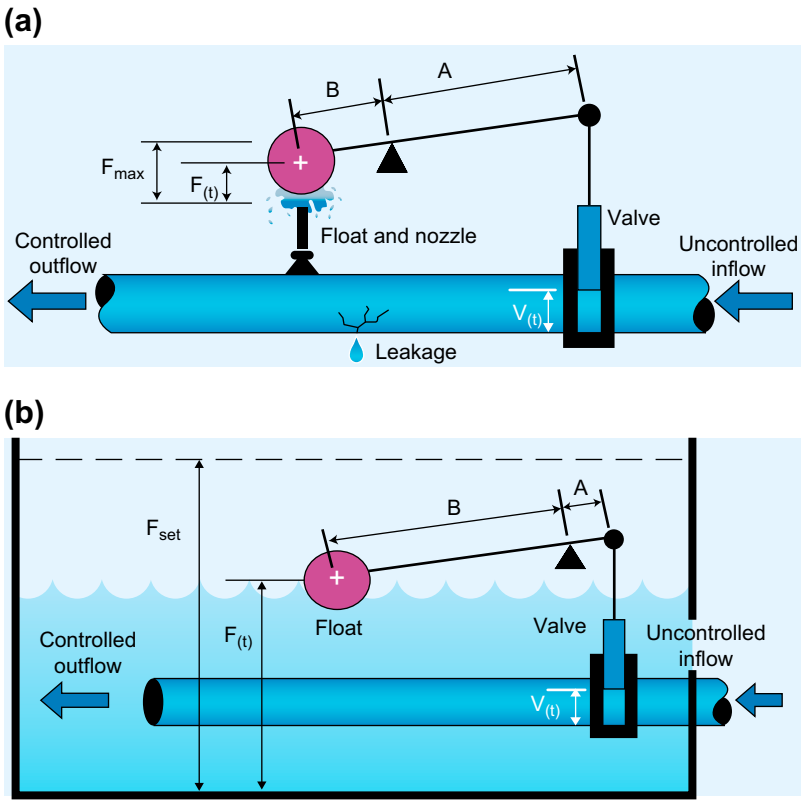


FIGURE 8.2

Two examples of PID control implementation. (a) Flow control: a mechanical flow controller manipulates the valve to maintain the downstream flow rate against leakage; (b) pressure control: a mechanical flow controller manipulates the valve to shut off the flow once the tank has filled to the desired level.

(Courtesy of Vance J. VanDoren; 2000.)

Equation (1) is an example of a process model that quantifies the relationship between the controller's efforts and its effects on the process variable. The controller also has a gain G_c, which determines its output at time t according to:

$$V(t) = G_c\,(F_{max} - F(t)) \tag{2}$$

The constant F_{max} is the highest possible float position, achieved when the valve's piston is completely depressed. The quantity $(F_{max} - F(t))$ that enters the controller as an input gains strength by a factor of A/B before it is output to the process as a control effort V(t). Note that controller Equation (2) can also be expressed as:

$$V(t) = G_c\,(F_{set} - F(t)) + V_B \tag{3}$$

where F_{set} is the desired float position (achieved when the flow rate equals the set-point) and $V_B = G_c (F_{max} - F_{set})$ is a constant known as the bias. A controller's bias represents the control effort required to maintain the process variable at its set-point in the absence of a load.

(1) Proportional control

Equation (3) shows how this simple mechanical controller computes its output as a result of the error between the process variable and the set-point. It is a proportional controller because its output changes in proportion to a change in the measured error. The greater the error, the greater the control effort; and as long as the error remains, the controller will continue to try to generate a corrective effort.

A possible problem in the example shown in Figure 8.2(a) is that a proportional controller tends to settle on the wrong corrective effort. As a result, it will generally leave a steady-state error (offset) between the set-point and the process variable after it has finished responding to a set-point change or a load. Suppose the process gain G_p is 1, so that any valve position V(t) will cause an identical float position F(t). Suppose also the controller gain G_c is 1 and the controller's bias V_B is 1. If the set-point for the flow rate requires F_{set} to be 3 inches and the actual float position is only 2 inches, there will be an error of $(F_{set} - F(t)) = 1$ inch. The controller will amplify that 1 inch error to a 2 inch valve opening according to Equation (3). However, since that 2 inch valve opening will in turn cause the float position to remain at 2 inches, the controller will make no further change to its output and the error will remain at 1 inch.

(2) Integral control

One of the first solutions to overcome this problem was integral control. An integral controller generates a corrective effort proportional, not to the present error, but to the sum of all previous errors. The level controller depicted in Figure 8.2(b) illustrates this point. It is essentially the same float-and-lever mechanism from the flow control example (see Figure 8.2(a)) except that it is now surrounded by a tank, and the float no longer hovers over a nozzle but rests on the surface of the water. This arrangement should look familiar to anyone who has inspected the workings of a common household toilet.

As in the first example (see Figure 8.2(a)), the controller uses the valve to control the flow rate of the water. However, its new objective is to refill the tank to a specified level whenever a load (i.e., a flush) empties the tank. The float position F(t) still serves as the process variable, but it now represents the level of the water in the tank, rather than the water's flow rate. The set-point, F_{set}, is the level at which the tank is full.

The process model is no longer a simple gain Equation like (1), since the water level is proportional to the total volume of water that has passed through the valve. That is:

$$F(t) = \text{Integral}\,(G_p\, V(t)\, dt) \tag{4}$$

Equation (4) shows that tank level F(t) depends not only on the size of the valve opening V(t) but also on how long the valve has been open.

The controller itself is the same, but the addition of the integral action in the process makes it more effective. Specifically, a controller that contains its own integral action, or acts on a process with inherent integral action will generally not permit a steady-state error. This phenomenon becomes apparent in this example. The water level in the tank will continue to rise until the tank is full and the valve shuts off. On the other hand, if both the controller and the process happened to be pure integrators as in Equation (4), the tank would overflow because back-to-back integrators in a closed loop cause the steady-state error to grow without bound.

(3) Derivative control

The basic idea of derivative control is to generate one large corrective effort immediately after a load change in order to begin eliminating the error as quickly as possible. The strip chart in Figure 8.3 shows how a derivative controller achieves this. At time t_1, the error, shown in the upper trace, has increased abruptly because a load on the process has dramatically changed the process variable. The derivative controller generates a control action proportional to the time derivative of the error signal.

The derivative of the error signal is shown in the lower trace which could be given by Equation (5).

$$F(t) = \text{Derivative } (F_{set} - F(t)) \tag{5}$$

Note the spike at time t_1. This happens because the derivative of a rapidly increasing step-like function is itself an even more rapidly increasing impulse function. However, since the error signal is much more level after time t_1, the derivative of the error returns to roughly zero thereafter. Derivative control action is zero when the error is constant and spikes dramatically when the error changes abruptly.

In many cases, adding this "kick" to the controller's output solves the performance problem nicely. The derivative action does not produce a particularly precise corrective effort, but it generally gets the process moving in the right direction much faster than a PI controller would.

(4) Combined PID control

A combined PID controller is able to perform these three control mechanisms automatically: Proportional control + integral control + derivative control. An illustration of a system using PID control is shown in Figure 8.4. This system needs a precise oil output flow rate controlled by pump motor speed. This is controlled through a control panel consisting of a variable-speed drive. In turn, the drive's speed control output is controlled by an electronic controller, whose output to the drive is determined by two factors.

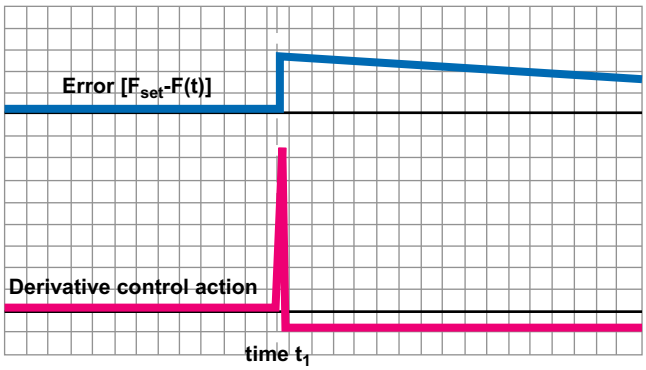


FIGURE 8.3

Derivative control example for the flow and pressure PID control implementation. The upper trace on this strip chart shows the error between the process variable F(t) and its desired value F_{set}. The derivative control action in lower trace is the time derivative of this difference.

(Courtesy of Vance J. VanDoren; 2000.)

The first factor is the set-point, determined by a dial setting (or equivalent device). Second, a flow sensor feeds back the actual output flow rate to the electronic controller. The controller compares the set point and the actual flow. If they differ for some reason, a corrective signal is sent to the motor controller, which changes motor speed accordingly, by changing the voltage applied to the motor. For example, if the output oil flow rate goes below the set point, a signal to speed up the motor is sent. The controller then uses PID control to make the correction promptly and accurately to return to the set point flow. If the dial is changed to a new setting, the function of the PID system is to reach the new set-point as quickly and accurately as possible.

If interested, a reader can analyze this example in Figure 8.4 to explain what the proportional control is, what the integral control is, and what the derivative control is.

8.1.3 PID controller tuning rules

Over the past decades, PID controllers have been very popular in industry due the simplicity of this control law and the small number of tuning parameters needed for PID controllers. Hundreds of tools,

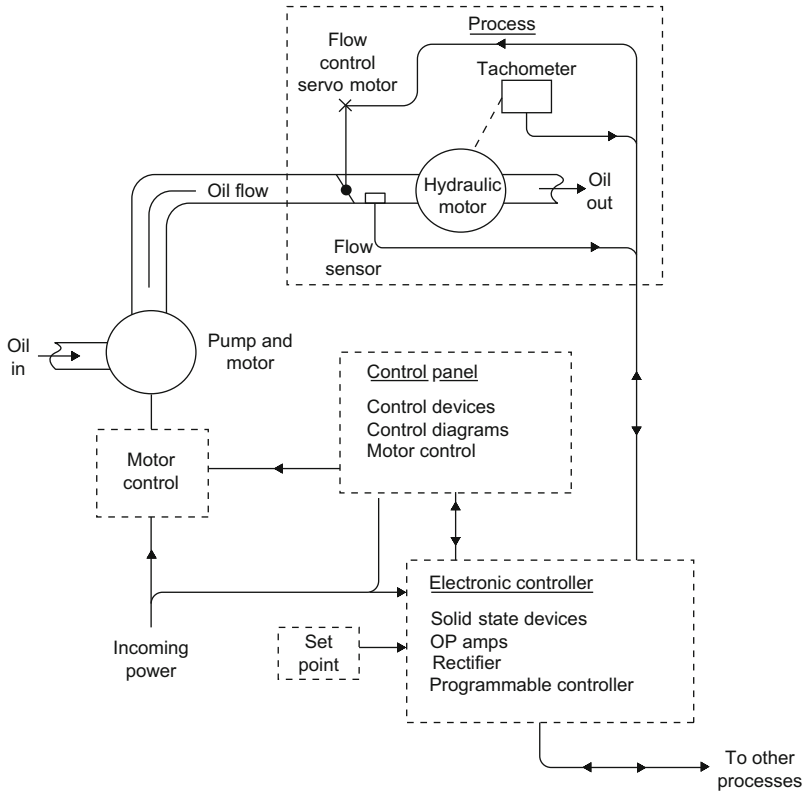


FIGURE 8.4

A combined PID control example for the control system of a hydraulic pump.

methods and theories are available for this purpose. However, finding appropriate parameters for PID controllers is still a difficult task, so in practice control engineers still often use trial and error for the tuning process.

There are many rules, sometimes called methods, for the proper tuning of PID controls, most of which require a considerable amount of trial and error as well as a technician endowed with a lot of patience. This technique will not solve every problem, but it has worked for many. This subsection describes two basic methods for tuning PID controllers to help readers get started.

(1) Ziegler Nichols tuning rules

There are two common techniques, the process reaction curve technique and the closed-loop cycling method, for tuning PID controllers. These two methods were first formally described in an article by J.G. Ziegler and N.B. Nichols in 1942.

Figure 8.5 describes the process reaction curve technique. It should be understood that optimal tuning, as defined by J.G. Ziegler and N.B. Nichols, is achieved when the system responds to a perturbation with a 4:1 decay ratio. That is to say that, for example, given an initial perturbation of $+40°$, the controller's subsequent response would yield an undershoot of $-40°$ followed by an overshoot of $+2.5°$. This definition of optimal tuning may not suit every application, so the trade-offs must be understood.

In the closed-loop cycling method, estimate the ultimate gain K_u and ultimate period T_u when a proportional part of the controller is acting alone. Follow these steps:

(a) Monitor the temperature response curve over time using an oscilloscope.
(b) Set the proportional gain low enough to prevent any oscillation in the response (temperature). Record the offset value for each gain setting.

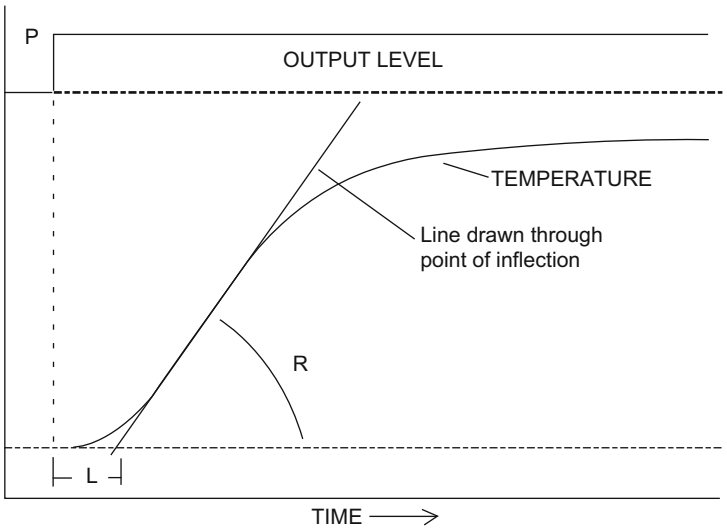


FIGURE 8.5

PID control process reaction curve.

(c) Increase the gain in steps of 2× the previous gain. After each increase, if there is no oscillation, change the set-point slightly to trigger an oscillation.
(d) Adjust the gain so that the oscillation is sustained, that is, continues at the same amplitude. If the oscillation increases, decrease the gain slightly. If it is decreasing, increase the gain slightly.
(e) Make note of the gain that causes sustained oscillations and the period of oscillation. These are the ultimate gain K_u and the ultimate period T_u, respectively.

(2) Tuning an ON OFF control system

If the controller contains integral and derivative adjustments, tune them to zero before adjusting the proportional band. This adjustment selects the response speed (sometimes called gain) a proportional controller requires to achieve stability in the system, and must be wider in degrees than the normal oscillations of the system but not so wide as to dampen the system response. Start out with the narrowest setting for the proportional band. If there are oscillations, slowly increase the proportional band in small increments, allowing the system to settle out for a few minutes after each step adjustment until the point at which the offset droop begins to increase. At this point, the process variable should be in a state of equilibrium at some point under the set-point. The next step is to tune the integral or reset action. If the controller has a manual reset adjustment, simply adjust the reset until the process droop is eliminated. The problem with manual reset adjustments is that once the set-point is changed to a value other than the original, the droop will probably return and the reset will once again need to be adjusted.

If the control has automatic reset, the reset adjustment adjusts the auto reset time constant (repeats per minute). The initial setting should be at the lowest number of repeats per minute to allow for equilibrium in the system. In other words, adjust the auto reset in small steps, allowing the system to settle after each step, until minor oscillations begin to occur. Then back off on the adjustment to the point at which the oscillations stop and the equilibrium is reestablished. The system will then automatically adjust for offset errors (droop). The last control parameter to adjust is the rate or derivative function. It is crucial to remember to always adjust this function last, because if the rate adjustment is turned on before the reset adjustment is made, the reset will be pulled out of adjustment when the rate adjustment is turned on. The tuning procedure is then complete.

The function of the rate adjustment is to reduce any overshoot as much as possible. The rate adjustment is a time-based adjustment measured in minutes, which is tuned to work with the overall system response time. The initial rate adjustment should be the minimum number of minutes possible. Increase the adjustment in very small increments. After each adjustment let it settle down for a few minutes. Then increase the set-point by a moderate amount. Watch the control action as the set-point is reached. If an overshoot occurs, increase the rate adjustment by another small amount and repeat the procedure until the overshoot is eliminated. Sometimes the system will become sluggish and never reach set-point at all. If this occurs, decrease the rate adjustment until the process reaches set-point. There may still be a slight overshoot but this is a trade-off situation.

8.1.4 PID controller software design

There is no specific way a PID should be implemented in firmware; the methods described here only touch on a few of the many possibilities. The PID routine is configured in a manner that makes it

modular. It is intended to be plugged into an existing piece of firmware, where the PID routine is passed the 8-bit or 16-bit or 32-bit error value that equals the difference between the desired plant response and the measured plant response. The actual error value is calculated outside the PID routine, but if necessary, the code could be easily modified to do this calculation within it. The PID can be configured to receive the error in one of two ways, either as a percentage with a range of 0 100% (8-bit), or a range of 0 4000 (16-bit) or even higher (32-bit). This option is configured by a #define statement at the top of the PID source code with the PID's variable declarations, if the programming language used is C or C++. The gains for proportional, integral, and derivative all have a range of 0 15. For resolution purposes, the gains are scaled by a factor of 16 with an 8-bit maximum of 255. A general flow showing how the PID routine would be implemented in the main application code is presented in Figure 8.6.

There are two methods for handling the signed numbers. The first method is to use signed mathematical routines to handle all of the PID calculations. The second is to use unsigned mathematical routines and maintain a sign bit in a status register. If the latter is implemented, there are five variables that require a sign bit to be maintained: (1) error, (2) a error,(3) p error, (4) d error, (5) pid error. All of these sign bits are maintained in the register defined as, for example, pid stat1 register.

Flowcharts for the PID main routine and the PID Interrupt service routine functions are shown in Figures 8.7 and 8.8, respectively. The PID main routine is intended to be called from the main application code that updates the error variable, as well as the pid stat1 error sign bit. Once in the PID main routine, the PID value will be calculated and put into the pid out variable, with its sign bit in pid stat1. The value in pid out is converted by the application code to the correct value so that it can be applied to the plant. The PID interrupt service routine is configured for a high priority interrupt. The

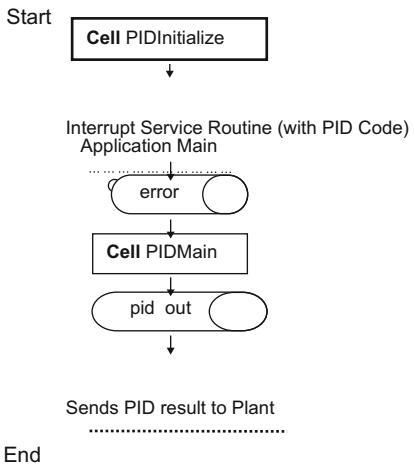


FIGURE 8.6

PID firmware implementation.

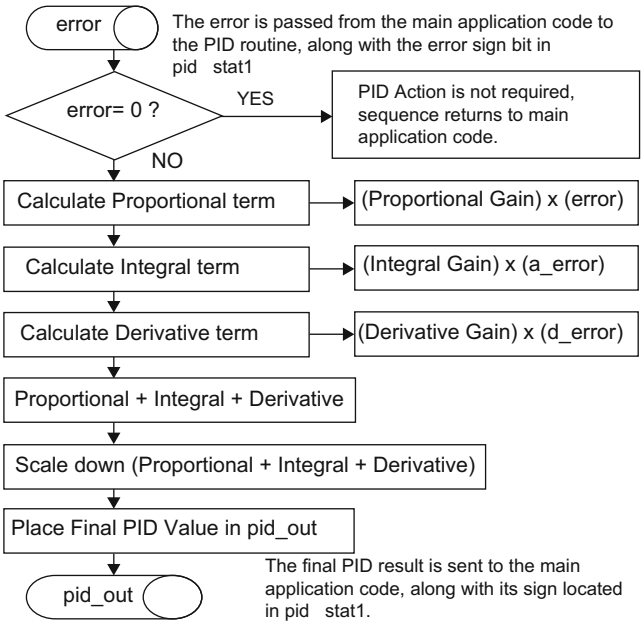
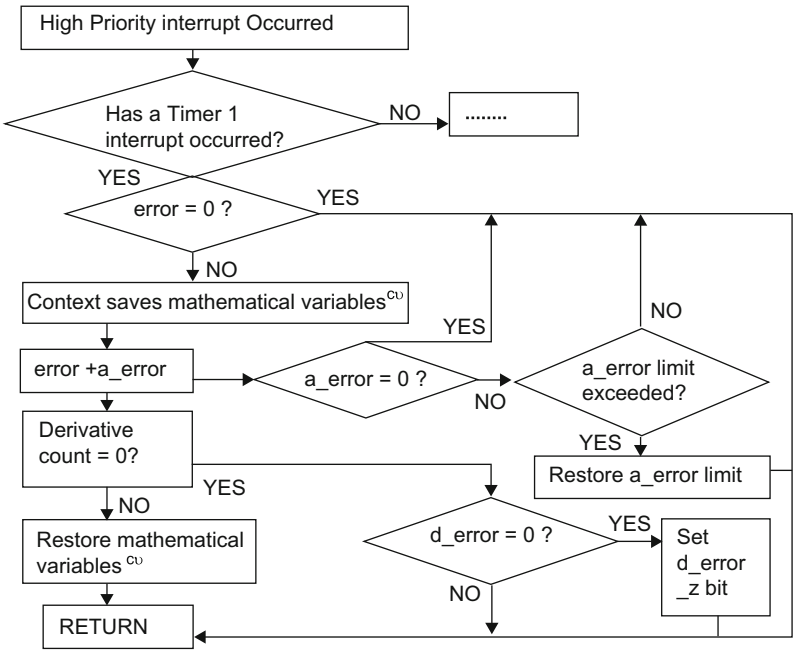


FIGURE 8.7

Main PID routine flow chart.

instructions within this interrupt service routine can be placed into an existing interrupt service routine, or kept as is and plugged into the application code.

The proportional is the simplest term. The error is multiplied by the proportional gain: (error) $\times$ (proportional gain). The result is stored in the variable, prop. This value will be used later in the code to calculate the overall value needed to go to the plant. To obtain the integral term, the accumulated error must be retrieved. The accumulated error, a error, is the sum of past errors. For this reason, the integral is known for looking at a system's history for correction. The derivative term is calculated in similar fashion to the integral term. Considering that the derivative term is based on the rate at which the system is changing, the derivative routine calculates d error. This is the difference between the current error and the previous error. The rate at which this calculation takes place is dependent upon the Timer1 overflow. The derivative term can be extremely aggressive when it is acting on the error of the system. An alternative to this is to calculate the derivative term from the output of the system and not the error. The error will be used here. To keep the derivative term from being too aggressive, a derivative counter variable has been installed. This variable allows d error to be calculated once for an x number of Timer1 overflows (unlike the accumulated error, which is calculated every Timer1 overflow). To get the derivative term, the previous error is subtracted from the current error (d error = error $-$ p error). The difference is then multiplied by the derivative gain and this result is placed in the variable, deriv, which is added to the proportional and integral terms.



CU These instructions are options; they are dependent upon how the Interrupt Service Routine is configured.

FIGURE 8.8

Flow chart of PID interrupt service routine.

8.2 BPC (BATCH PROCESS CONTROL) CONTROLLERS

Batch process control (or batching control) is primarily used for repeated filling operations that lead to the production or the transport of finite quantities of material, by subjecting quantities of input materials to an ordered set of processing activities over a finite period and using one or more pieces of equipment. Batch control systems are also engineered to control, measure, and dispense any volume of liquid from drums, tanks, or other large storage vessel. Generally, they are used in any industry where batching, chemical packaging, or dilution is required to be accurate and efficient. For example, in the pharmaceutical and consumer products industries, batch control systems provide complete automation of batch processes including the mixing, blending, and reacting of products such as spice blends, dairy products, beverages, and personal care products.

Typical batch applications include chemical packaging, waste-water treatment, machine manufacturing, pharmaceutical filling, and industrial plating. Figure 8.9 gives some examples of typical applications of batch control. As shown in the first example, the batch controller is ideal for discrete manufacturing as well as repetitive fill operations. Here, the batch controller counts bottles that it then groups into six-packs. Its control capability can be used to track bottles or six-packs. The second example is a drum-filling application, in which the batch controller utilizes its maximum of

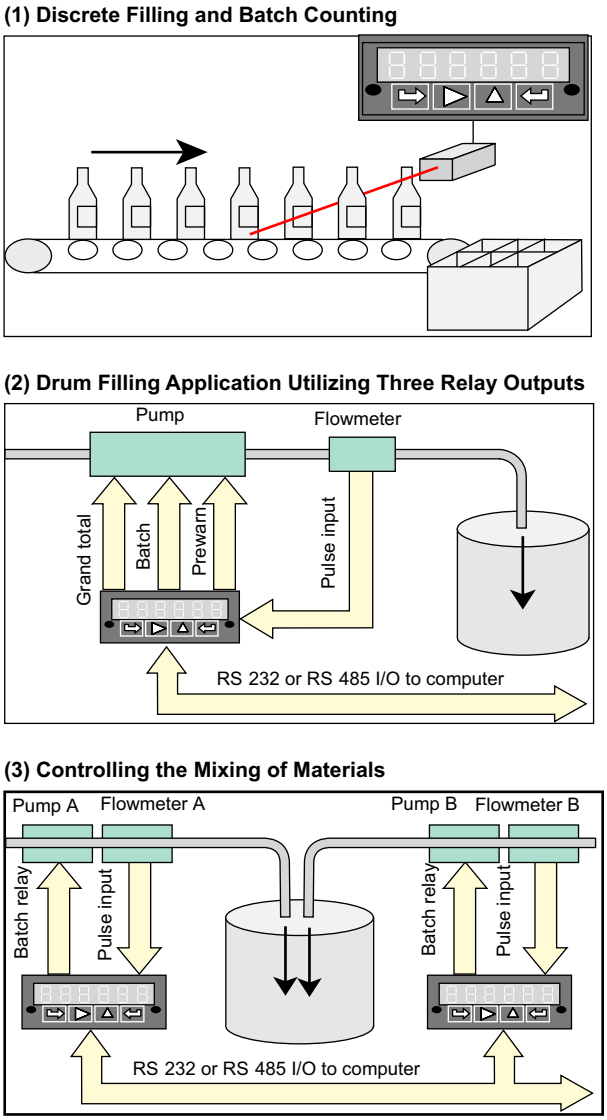


FIGURE 8.9

Some examples of typical applications of the batch process control.

three relays to control the pump. The prewarn relay slows down the pump near the preset to avoid overshoot. The batch relay stops the pump at the preset. The controller relay stops the filling operation once a predetermined number of drums have been filled. The third is for multiple-batch controllers that can be used in combination to control the mixing of materials in the proper ratio. Each feed line is equipped with its own pump, flow meter, and Laureate. Controller setup and monitoring of the mixing

operation are facilitated by optional serial communications. An RS-232 or RS-485 I/O Interface allows a single data line to handle multiple controllers.

8.2.1 Batch control standards

Before the advent of industry-accepted standards for batch processing, every batch facility operated in its own world. Even something as basic as nomenclature was different in every plant. What might be called a "run" in one facility might be called a "batch" in another. What was a "formula" to one manager could be a "recipe" to another. Entire process methodologies varied just as widely. There were as many different ways to structure and run a batch as there were facilities in the world.

In 1995, ISA (The Instrumentation, Systems, and Automation Society) approved one of its first standards for batch processing, called ISA-88, or simply S88. These standards provided a framework for the design and operation of batch facilities, specifying terminology, process models, data structures and more.

Another standard, ISA-95 or S95, was developed in 2000. This standard is an outgrowth of the S88 standard, and commonly referred to as an international standard for developing an automated interface between enterprise and control systems.

The S88 and S95 standards have been widely accepted by the batch manufacturing community worldwide, which has allowed enterprises to improve communication, promote modularity, reduce maintenance costs and downtime, and improve overall performance. The savings can be substantial according to the ISA, plants using batch processing standards can save up to 30% of the cost of designing and implementing a batch processing system from scratch, and save domestic operations up to 15% of the cost of meeting the criteria for automation reliability.

(1) ANSI/ISA-88

ISA-88, or S88, is a standard that addresses batch process control that was approved by the ISA in 1995. It was adopted by the IEC in 1997 as IEC 61512-1. It is a design philosophy for describing equipment and procedures, and is equally applicable to manual processes. The current parts of the S88 standard include;

1. ANSI/ISA-88.01-1995 Batch Control Part 1: Models and terminology.
2. ANSI/ISA-88.00.02-2001 Batch Control Part 2: Data structures and guidelines for languages.
3. ANSI/ISA-88.00.03-2003 Batch Control Part 3: General and site recipe models and representation.
4. ANSI/ISA-88.00.04-2006 Batch Control Part 4: Batch Production Records.

S88 provides a consistent set of standards and terminology for batch control and defines the physical model, process model procedures, and recipes. Each of the processes in the model consists of an ordered set of stages which in turn consists of an ordered set of process operations made up of an ordered set of actions.

The physical model begins with the enterprise, which must contain a site which may contain areas which may contain process cells which must contain a unit which may contain equipment modules which may contain control modules. Some of these levels may be excluded, but not the unit.

The procedural control model consists of recipe procedures which consist of an ordered set of unit procedures which consist of an ordered set of operations which consist of an ordered set of phases. Some of these levels may be excluded.

Another fundamental issue was to separate recipes from equipment or equipment-independent recipes. This allowed the creation of agile and flexible batch manufacturing plants, where modularity and reuse of the recipes and elements that made up the plant equipment model could be defined and further developed into objects and libraries. Recipes could be either general, site, master, or control. The contents of the recipe include; header, formula, equipment requirements, procedure, and other information.

(2) ANSI/ISA-95

ISA-95 or S95 is an international standard for developing an automated interface between enterprise and control systems. This standard has been developed for global manufacturers, and can be applied in all industries, and in all sorts of processes, such as batch processes, continuous and repetitive processes. The objectives of ISA-95 are to provide consistent terminology for supplier and manufacturer by means of consistent information and operations models, which are a foundation for clarifying application functionality and how information is to be used.

Within production areas, activities are executed and information is passed back and forth. The standard provides reference models for production activities, quality control activities, maintenance activities and inventory activities.

There are five parts of the ISA-95 standard, listed below:

1. ANSI/ISA-95.00.01-2000, Enterprise-Control System Integration Part 1: Models and Terminology consists of standard terminology and object models, which can be used to decide the information which should be exchanged.
2. ANSI/ISA-95.00.02-2001, Enterprise-Control System Integration Part 2: Object Model Attributes consists of attributes for every object that is defined in part 1. The objects and attributes of part 2 can be used for the exchange of information between different systems, but these objects and attributes can also be used as the basis for relational databases.
3. ANSI/ISA-95.00.03-2005, Enterprise-Control System Integration, Part 3: Models of Manufacturing Operations Management focuses on the functions and activities at level 3 (Production / MES layer; MES: Manufacture Execution System). It provides guidelines for describing and comparing the production levels of different sites in a standardized way.
4. ISA-95.04 Object Models & Attributes Part 4 of ISA-95: Object models and attributes for Manufacturing Operations Management. This technical specification defines object models that determine which information is exchanged between MES activities (which are defined in part 3 of ISA-95). The models and attributes from part 4 are the basis for the design and the implementation of interface standards and ensure flexible cooperation and information-exchange between the different MES activities.
5. ISA-95.05 B2M Transactions Part 5 of ISA-95: Business to Manufacturing Transactions. This technical specification defines operation between office and production automation systems, which can be used together with the object models of parts 1 and 2. The operations connect and organize the production objects and activities that are defined in earlier parts of the standard. Such operations take place on all levels within a business, but the focus of this technical specification lies on the interface between enterprise and control systems. On the

basis of models, the operation will be described and the operation processing is logically explained.

8.2.2 Batch control systems

Most batch control systems provide automated process controls, including recipe development, batch management and execution, production scheduling, equipment performance monitoring, production history and material tracking functionalities. However, some batch control systems are also designed to improve batch scheduling, batch consistency and the automatic collection of real-time, reliable and accurate batch event data. This is because industry now needs to produce smaller batches of products with greater variation, quickly and efficiently. There are a range of batch control systems, designed and implemented using the ISA-88 standards and optimized using ISA-95 standards, to produce product batches repeatedly and consistently with minimum costs.

The architectures of batch control systems depend upon their physical structures. There are three classes of physical structure approved in the ANS/ISA-88.01-1995:

1. A single-path structure is a group of units through which a batch passes sequentially (see Figure 8.10(a)). A single-path structure could be a single unit, such as a reactor, or several units in sequence. Multiple input materials are typically used; multiple finished materials may be generated. Several batches may be in progress at the same time.
2. A multiple-path structure is shown in Figure 8.10(b). It consists of multiple single-path structures in parallel with no product transfer between them. The units may share raw material sources and product storage. Several batches may be in progress at the same time. Although units within a multiple-path structure may be physically similar, it is possible to have paths and units within a multipath structure that are of radically different physical design.
3. A network structure is shown in Figure 8.10(c). The paths may be either fixed or variable. When the paths are fixed, the same units are used in the same sequence. When the path is variable, the sequence may be determined at the beginning of the batch or it may be determined as the batch is being produced. The path could also be totally flexible. For example, a batch would not have to start at either Unit 1 or Unit 3; it could start with any unit and take multiple paths through the process cell. The units themselves may be portable within the process cell. In this case, verification of the process connections may be an important part of the procedures. Note that several batches may be in production at the same time. The units may share raw material sources and product storage.

A typical batch control system, regardless of physical structure, consists of the following components:

1. Field devices, such as sensors, actuators and instrumentation, that provide the batch control system with information and allow it to control the production process.
2. Batch controllers, such as special PLC, CNC, industrial computers or software packages, that receive the signals from the field devices and processes them according to pre-defined sequences in order to control actuators such as pumps, valves and motors.
3. Human machine interface as a system that allows interaction with the operators in the plant to monitor the status of the batch control system, to issue commands and to modify parameters.
4. Batch application, which visualizes and develops recipes.

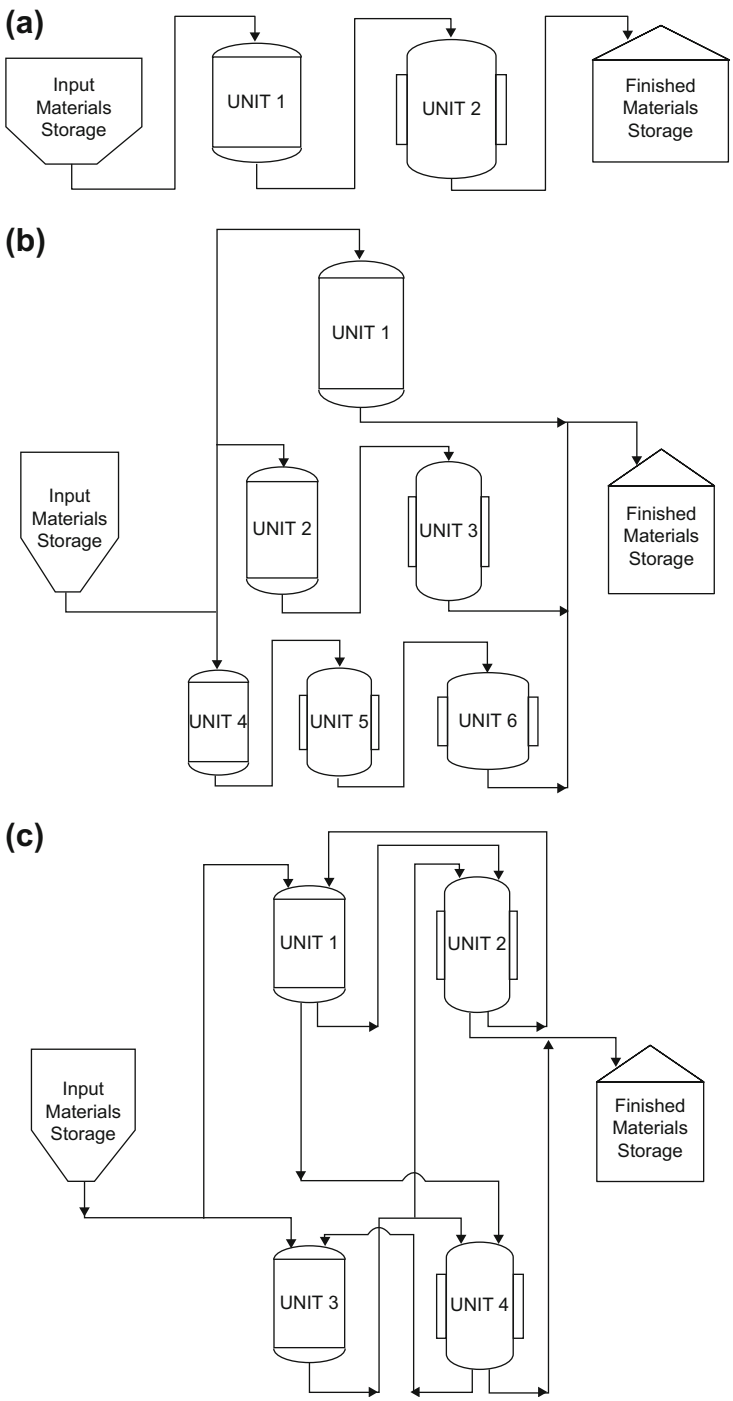


FIGURE 8.10

The batch control classification by physical structure. (a) Single-path structure; (b) multiple-path structure; (c) network structure.

(Courtesy of the ISA, 1995.)

With these components, all batch process control systems are capable of performing the following functions:

1. There must be an I/O interface between the processes and the control. The I/O interface passes all the information moving between the two parts of the total system.
2. The control system must provide continuous control functions to regulate appropriate portions of the process.
3. The control system must provide sequential control functions. A properly designed batch control system should make it easy to describe the sequence of operations and the checks that must take place.
4. The control system must provide displays and interfaces which the operator can use to monitor and direct process activity.

In particular, if a batch control system is digital it should have additional functions:

1. A digital control system that allows the process to handle a range of products, and not just a single one. The use of recipes is a concise and convenient method to describe the process steps for each product.
2. An advanced control system that provides a multiprogramming environment in which each specific task can be programmed in a simple stand-alone fashion. The system should also allow more than one unit in the plant to be controlled at the same time.
3. An advanced control system that provide displays which are oriented toward the total process rather than toward individual parameters.
4. An advanced control system that automatically detects and corrects process upsets and equipment failures.
5. An advanced control system that provides for device operations which reduce the complexity of the logic that the user must deal with. There are many mundane and repeated operations and error-checking functions which must be done where manipulating valves, pumps, motors, fans, and so on which can best be handled in device packages which standardize such operations.

8.2.3 Batch control mechanism

The mechanism for batch control depends on two factors; managing the batch processes, and handling recipes and batches.

(1) Managing batch processes

Batch processes deal with discrete quantities of raw materials or products. They allow more than one type of product to be processed simultaneously, as long as the products are separated by the equipment layout. They entail movement of discrete products from processing area to processing area. They have recipes (or processing instructions) associated with each load of raw material to be processed into product. They have more complex logic associated with processing than is found in continuous processes, and often include normal steps that can fail, and thus also include special steps to be taken in the event of a failure, which therefore gives exception handling in batch processes great importance.

Each step can be simple or complex in nature, consisting of one or more operations. Generally, once a step is started it must be completed to be successful. It is not uncommon to require some operator approval before leaving one step and starting the next. There is frequently provision for non-normal exits to be taken because of operator intervention, equipment failure, or the detection of hazardous conditions. Depending on the recipe for the product being processed, a step may be bypassed for some

products. The processing operations for each step are generally under recipe control, but may be modified by operator override action. A typical process step is shown in Figure 8.11.

(2) Handling recipes and batches

Recipe and batch capability are standard requirements in many industries and applications. All batch control systems compatible with ISA-88 and ISA-95 standards should fully support batch and recipe handling, including recipe storage; automatic form filling; automatic batch creation and monitoring; and proof of process reporting. To perform the recipe and batch handling, the facilities below can be very helpful.

(1) Recipe data

Recipe data are a collection of control set-points that define the parameters required to make a specific product or to control a specific process. This handling allows any number of such recipes to be created, edited and called up via user-defined forms (such as in Microsoft Access format). These forms may display the recipe data and optionally allow items to be modified. They may also allow additional data to be entered that do not form part of the recipe but which may be required as part of the batch history.

(2) Batch data handler

The batch data handler is a database handler that makes it easy to handle the requirements of a batch process without the need to write programs or configuration scripts. The handler works by monitoring the system and watching for the occurrence of any of the criteria that have been defined as meaning that a batch is beginning. When the batch begins, it can reset selected signals and also store the values of any number of signals. As the batch progresses, the values of any number of

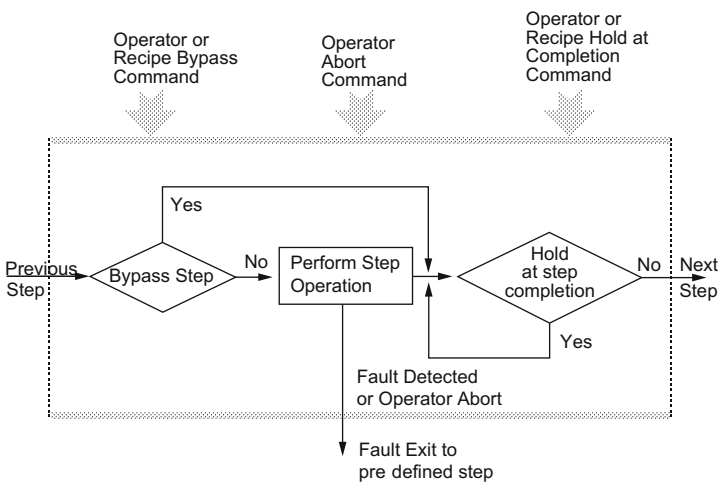


FIGURE 8.11

A typical step of batch process control.

signals may be periodically written into the batch database, to keep a working record of how the batch is progressing. A batch ends when the conditions defining the end of the batch become true, such as a digital value going low, or a given expression evaluating to true. Again, data may be written to the batch database and signal values reset when the batch ends. It can also perform statistical surveys, calculating such values as mean, max, min, standard deviation, runtime, number of operations etc.

(3) Batch based trend recorder

In recipe and batch handling, all relevant parts of batches can be accessed from within historical trends where the start and end dates shown on the trend will be taken from the batch that has been selected. A tag substitution facility means that the same trend can be used to show data from different batches, even if those batches relate to different parts of the plant.

(4) Batch reporting

Proof of process reports may be generated automatically by a batch data handler or trend recorder when a batch ends. They may also be called up manually at any time. These reports, which are user-configurable, can display: (1) one off batch data such as Operator ID, batch run time, materials used, total product made etc.; (2) a list of all the alarms that occurred during the batch; (3) any number of trends each showing any number of signals. If there are too many data to display neatly on one batch report, the batch data recorder can be configured to run as many different reports as required.

In conclusion, the key points for recipe and batch handling are: (1) batches can be created manually or automatically; (2) facilities to continue an existing batch or start afresh; (3) any number of batches can be created; (4) full reporting, including multi-batch reporting, is dynamically feasible.

8.2.4 An integrated batch process controller

An integrated batch process controller, Simatic Batch, was developed by the Siemens AG in 2008. Simatic Batch is an industrial solution or batch control which; (1) is a high-level design solution able to integrate some distributed system (such as SCADA systems) into a batch control system; (2) it provides a batch control package as server/scheduler of the recipes, models, and tools compatible with the ISA-88.01 standard.

In fact, Simatic Batch plays a role as an integrated batch controller applied to a pre-defined PCS7 project with the required batch interface blocks and SFC blocks. Thus, each device in the control system is viewed as a complete object, that is, it is associated with a pre-defined set of I/O, is defined and controlled using CFC logic, and is visible on the SCADA as a single device with its alarms, real-time trends and information pop-ups. As such, once a project has been completed and commissioned, the user has complete control of the automation system without needing to modify the lower level control.

Simatic Batch provides special faceplates for controlling and monitoring plant units and equipment modules.

(1) Batch control center and batch planning

The batch control center is the "command center" for monitoring and controlling batch processes for Simatic Batch. Using the batch control center, it is possible to manage the data from a graphical user interface (see Figure 8.12(a)). It offers powerful functions including:

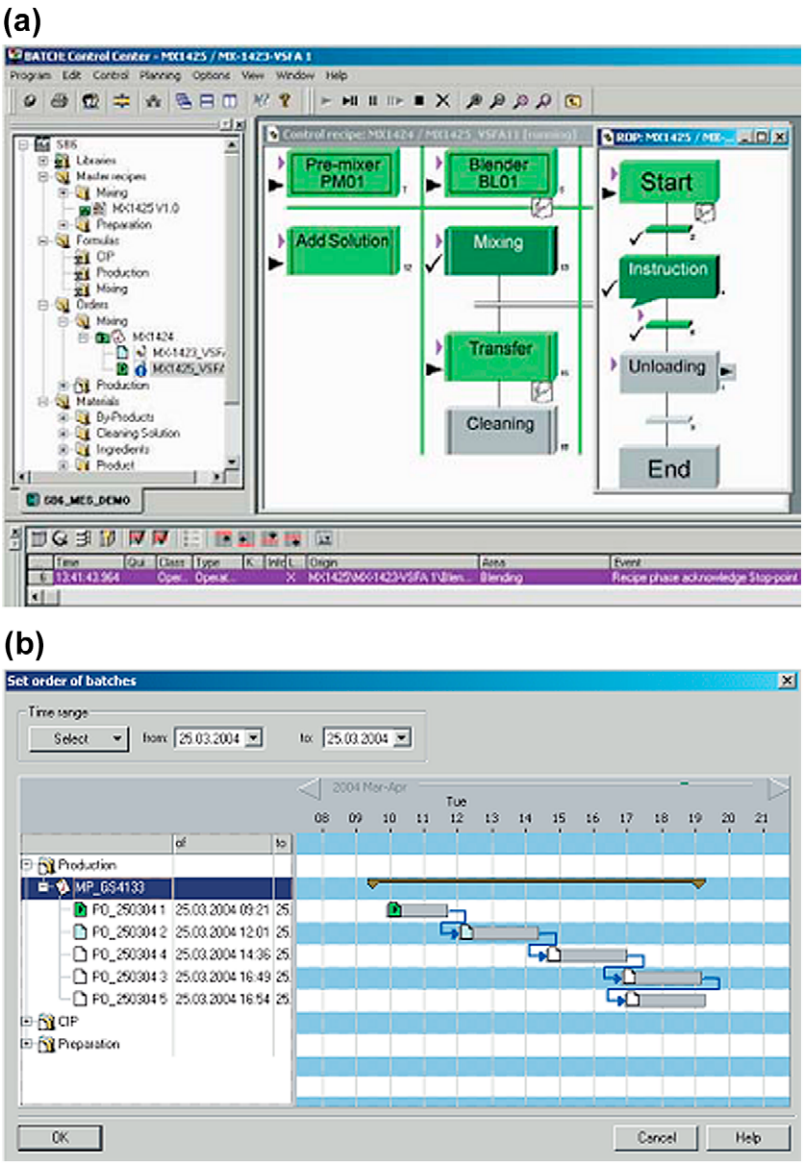


FIGURE 8.12

Simatic Batch: (a) a graphical display of batch control centre; (b) a graphic display of batch planning.

1. reading in and updating the plant data of the basic automation;
2. defining user privileges for all functions, for clients, or for plant units of Simatic Batch;
3. management of master recipes, and starting the recipe editor in order to enter the recipe structure;
4. exporting and importing of basic recipes, formulas and library objects;
5. creation of batches with master recipes; starting of batch processing and controlling of batches;
6. monitoring and diagnostics of batch processing and recording and archiving of recipes and batch data.

The batch control center enables the creation of individual production orders and batches. A greatly increased planning functionality is offered by the batch planning option package with which the batches can be planned in advance for a large number of production orders.

In addition to planning, the scope of functions include the modification, cancellation, deletion and enabling of batches. Creation and distribution of the batches for a production order are possible manually, but can also be carried out automatically depending on the definition of the batch number or production quantity. All batches and their occupation of plant units can be clearly presented in a combination of Gantt diagram and table, as shown in Figure 8.12(b). Time conflicts or conflicts resulting from multiple occupation of plant units are identified by symbols. Such conflicts can be eliminated simply by shifting the associated batches in the Gantt diagram.

(2) Recipe editor and batch report

The recipe editor is a user-friendly tool for the easy, intuitive creation and modification of basic recipes. It has a graphical user interface, as shown in Figure 8.13(a), processing functions for individual and grouped objects, and a structural syntax check. The basis for recipe creation is the batch objects created from the batch plant configuration using the Simatic PCS7 engineering system, e.g. plant units and technological functions. The following tasks can be performed with the recipe editor:

1. creation of new basic recipes and library operations;
2. querying of statuses of the recipe objects and of process values in transition conditions;
3. documentation of basic recipes;
4. selection of plant unit candidates through limitation of equipment properties;
5. releasing basic recipes for test or production;
6. configuring arithmetic expressions for calculating set-points for transitions and recipe parameters from recipe variables and constants.

The batch report function integrated in the recipe editor is used to produce recipe and batch reports. These reports (an example is given in Figure 8.13(b)) contain all data required for reproduction of the batch process, for proof of the quality, and for compliance with statutory directives. These include, for example: identification data; control recipe data; effective production data; time sequence of steps; status messages, fault messages and alarms; operator interventions; process values.

(3) Hierarchical and plant-unit neutral recipes

Simatic Batch provides a functional unit that fully covers the models described in the ISA-88.01 standard. The hierarchical recipe structure is mapped on the plant module as follows:

1. recipe procedure for controlling the process or the production in a plant;

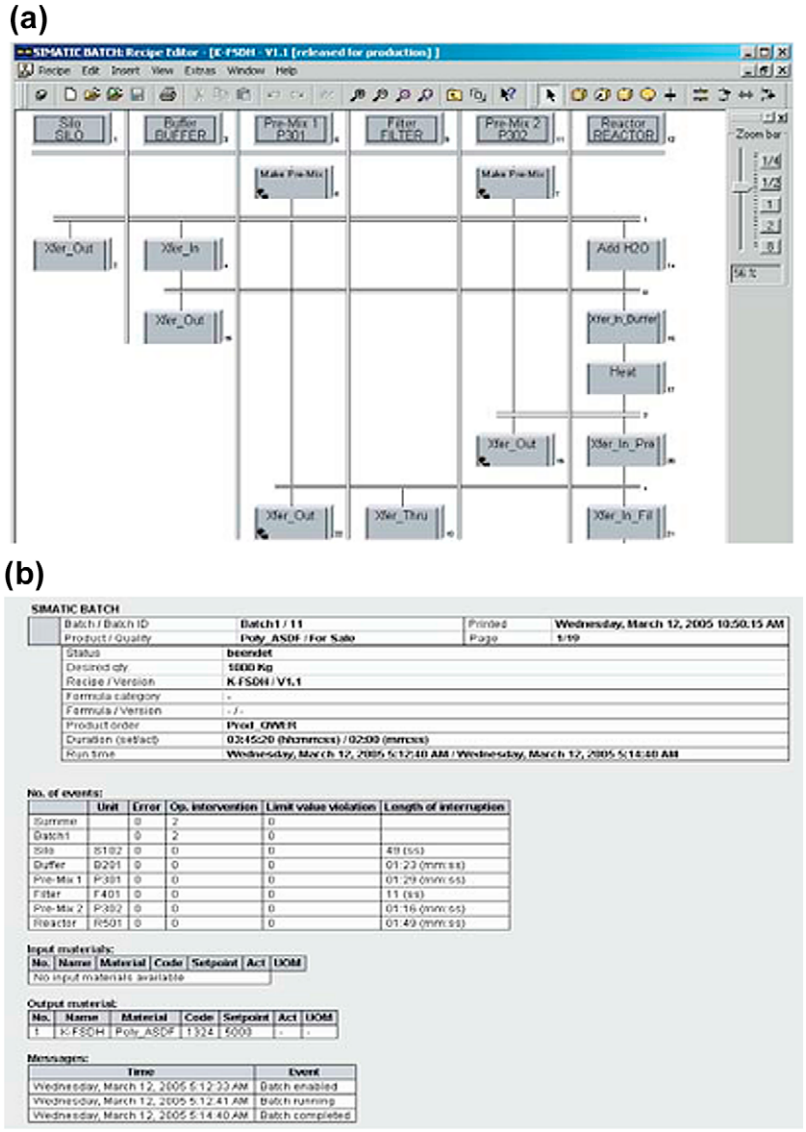


FIGURE 8.13

Simatic Batch: (a) a graphical display of the recipe editor; (b) a graphic display of a batch report.

2. partial recipe procedure for controlling a process step in a plant unit;
3. recipe operation/function for the process engineering task/function in an equipment module.

Creation of a recipe which is neutral to the plant unit minimizes the engineering overhead and provides significant advantages for validation. During creation of the recipe, the partial recipe procedures are

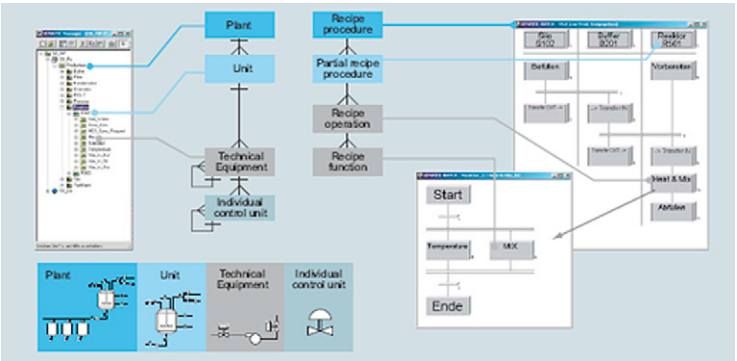


FIGURE 8.14

Simatic Batch: a graphical representation of hierarchical recipe structure in the plant model.

only assigned plant unit classes. The final assignment of plant units is only carried out at run-time. In the cases of batches which run for a longer period and where the plant units are not to be already determined and occupied at the start of a batch, the assignment is only carried out at the time of use. Conflicts in the occupation of plant units are detected by the system, and displayed. Figure 8.14 is a representation of hierarchical recipe structure in the plant model.

The following occupation strategies for plant unit assignments permit optimum orientation according to the special plant situation. It ought to noted that the occupation strategy can also be modified during the batch run-time, just like plant unit assignment.

1. "Manual selection of plant unit" for preselection at time of recipe creation.
2. "Preferred plant unit" for preselection at time of recipe creation.
3. Determination of "plant unused for longest time" to achieve uniform utilization.
4. Assignment of plant unit to be used by means of "process parameters" from an external module (e.g. scheduler).

8.3 SMC (SERVO MOTION CONTROL) CONTROLLERS

Servo control, which is also referred to as motion control or robotics, is used in industrial processes to move a specific load in a predetermined manner.

In practice, servo control can be used for solving two fundamental classes of motion control problems.

The first general class of problems deals with command tracking, which addresses the question of how well the actual motion follows what is being commanded. The typical commands for rotary motion control are position, velocity, acceleration and torque. For linear motion, force is used instead of torque.

The second general class of servo control addresses the disturbance rejection characteristics of the system. In this case, disturbances can be anything from torque disturbances on the motor shaft to incorrect motor parameter estimations. Disturbance rejection control reacts to unknown disturbances and modeling errors.

Complete servo control systems combine both these types of servo control to provide the best overall performance.

Servos are used in many applications all over the world, such as many remote-control devices, steering mechanisms in cars, wing control in airplanes, and robotic arm control in workshops. Servo control is manipulated by servo control devices, which provide all control information. A number of products have been commercially developed aimed at meeting the demands of precision servo applications, with new products currently being researched.

8.3.1 Servo control systems

Feedback systems typical are servo control systems used to control position, velocity, and/or acceleration. Figure 8.15 is a graphical representation of a typical servo control system. The controller and industrial digital drives contain the algorithms to close the desired loop (typically position or speed) and also handle machine interfacing with inputs/outputs, terminals, etc. The drive or amplifier closes another loop (typically speed or current) and represents the electrical power converter that drives the motor according to the controller reference signals. The motor can be DC or AC, rotary or linear. It is the actual electromagnetic actuator, generating the forces or torques required to move the load. Feedback elements such as tachometers, transmitters, encoders and resolvers are mounted on the motor and/or load in order to close the various servo loops.

(1) Controller

The controller is the brains of a servo control system. It is responsible for generating the motion paths and for reacting to changes in the outside environment. Typically, the controller sends a command signal to the drive; the drive provides power to the motor; and the feedback from the motor is sent back to the controller and drive. Feedback from the load is also routed to the controller. The controller analyzes the feedback signal and sends a new signal to the amplifier to correct for errors. The controller is considered to be the intelligent part of the servo, closing the speed and/or position loops while the amplifier closes the current loop, but may also close the speed and/or position loops, placing less demand on the controller.

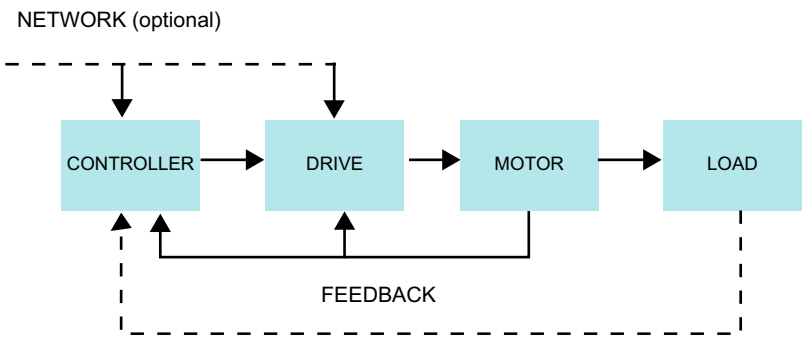


FIGURE 8.15

Typical servo control system: architecture and components.

Controllers come in a variety of forms, selected on the basis of cost, performance, convenience, and ease of use. Most controllers fall into the category of microcontrollers, which include PIC and R/C controllers, PLCs, and motion controllers.

In a servo system, the controller is the device that activates motion by providing a command signal to start, or change speed, or position, or both. This command is amplified and applied to the motor, and motion commences. Systems that assume the motion has taken place (or is in the process of taking place) are termed "open loop". An open loop drive (Figure 8.16(a)) is one in which the signal goes in only one direction from the control to the motor no signal returns from the motor or load to inform the control that action or motion has occurred.

If a signal is returned, then the system is described as having a signal that goes in two directions; the command signal goes out (to move the motor), and a signal is returned (the feedback) to the control to inform the control of what has occurred. The information flows back, or returns, and this is termed "closed loop" (Figure 8.16(b)).

The open loop approach is not good for applications with varying loads; it is possible for a stepper motor to lose steps; its energy efficiency level is low; and it has resonance areas that must be avoided. Applications that use the closed loop technique are those that require control over a variety of complex motion profiles. These may involve tight control of either speed and/or position; high resolution and accuracy; very slow, or very high velocity; and the application may demand high torques in a small package size.

(2) Drive

The servo drive is the link between the controller and motor. Also referred to as servo amplifiers, their job is to translate the low-energy reference signals from the controller into high-energy power signals to the motor. Originally, drives were simply the power stage that allowed a controller to drive a motor. They started out as single quadrant models that powered brushed motors. Later they incorporated four quadrant capabilities and the ability to power brushless motors. Four quadrants means the ability to both drive and regenerate from a motor in both directions.

Special servo drives and amplifiers are designed and manufactured for advanced motion control. They are used extensively where precise control of position and/or velocity is required. The drive or amplifier simply translates the low-energy reference signals from the controller into high-energy

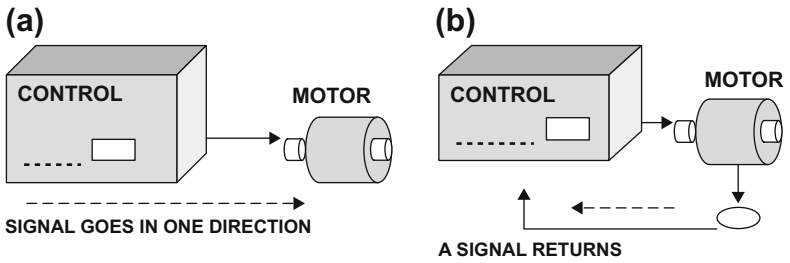


FIGURE 8.16

Servo control types: (a) open-loop drive; (b) closed-loop drive.

signals to provide motor voltage and current. In some cases a digital drive replaces either the controller/drive or controller/amplifier control system. The reference signals represent either a motor torque or a velocity command, and can be either analog or digital in nature.

The current trend is to add more features and abilities to drives. Today drives can be expected to handle all of the system feedback including encoders, resolvers and tachometers, as well as limit switches and other sensors. Drives are also being asked to close the torque loop, speed loop and position loop and being given the responsibility of path generation. As the line between controller and drive blurs, the drive will take on many of the more complex control functions that used to be the sole domain of the controller.

(3) Motor

The motor converts the current and voltage from the drive into mechanical motion. Most motors are rotary, but linear motors are also available. Many types of motors can be used in servo applications, as discussed in subsection 8.3.4.

(4) Load

Load considerations should include the object that is being moved, the moving parts in the machine and anything that may cause unwanted instabilities, such as couplings and backlash. The total mass of the moving parts in the machine all have inertias that the motor will need to overcome. Friction points such as from linear stages and bearings will add to the motor load. Flexible couplings will add resonances that have to be considered.

All servo systems consist of some kind of movement of a load. The method in which the load is moved is known as its motion profile. This can be as simple as a movement from point A to point B on a single axis, or as complex as bipedal stability control of a 26-axis humanoid robot. An example of a movement is shown in Figure 8.17. The y-axis represents the velocity (speed), and the x-axis represents time. The total distance travelled, D, is found by calculating the area under the curve. T is the total time required for the move. All motion profiles will require the load to accelerate and

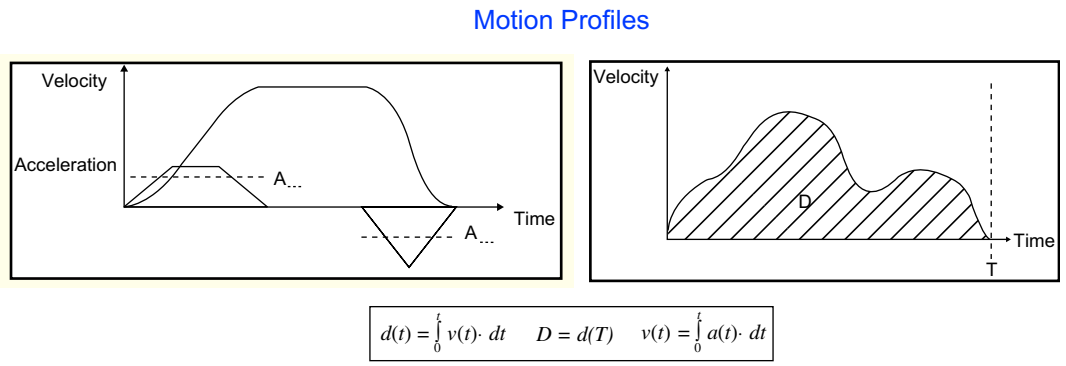


FIGURE 8.17

An illustration of the motion profiles and the formulas for servo motion control.

decelerate at least once during the move. In Figure 8.17, the slope of the velocity curve represents the acceleration or deceleration at that particular instant.

There are several types of motion profiles used with servo control systems. The most often used are constant speed, trapezoidal, and S-curve motion profiles.

(5) Feedback

In modern control systems, feedback devices are used to ensure that the motor or load reaches the commanded (required) position or speed. Servo amplifiers and controllers use this feedback to determine how much current to deliver to the motor at any given time, based on its present, actual position and speed compared with that required. There are two main types of feedback; absolute and relative (also known as incremental-only).

1. Absolute feedback: absolute devices provide definitive position within a specified range without any movement (without a homing routine).
2. Relative feedback (incremental): these devices provide only incremental position updates. In order to know the motor or load's position, this incremental feedback needs to be used in conjunction with some type of absolute feedback (a limit switch, for example).

Within these two general types of feedback, there are many different devices. Subsection 8.3.4 illustrates some of those most commonly used in motion controls.

8.3.2 Servo control mechanism

Servo control is the regulation of speed (velocity) and position of a motor based on a feedback signal. The most basic servo loop is the speed loop. This produces a torque command to minimize the error between speed command and speed feedback. Most servo systems require position control in addition to speed control, most commonly provided by adding a position loop in cascade or series with a speed loop. Sometimes a single PID position loop is used to provide position and speed control without an explicit velocity loop.

Servo loops have to be tuned (or compensated) for each application, by setting servo gains. Higher servo gains provide higher levels of performance, but they also move the system closer to instability. Low-pass filters are commonly used in series with the velocity loop to reduce high-frequency stability problems. Filters must be tuned at the same time as the servo loops. Some drive manufacturers deal with demanding applications by providing advanced control algorithms, which may be necessary because the mechanics of the system or the performance requirement do not allow the use of standard servo loops. In conclusion, servos are tuned or compensated through adjustments of gain and response so that the machine will produce accurate parts at a high productivity rate.

Motor control describes the process of producing actual torque in response to the torque command from the servo control loops. For brush motors, motor control is simply the control of current in motor winding, since the torque is approximately proportional to this current. Most industrial servo controllers rely on current loops, which are similar in structure to speed loops, but operate at much higher frequencies. A current loop takes a current command (usually just the output of the speed loop), compares it to a current feedback signal and generates an output that is essentially a voltage command. If the system needs more torque, the current loop responds by increasing the

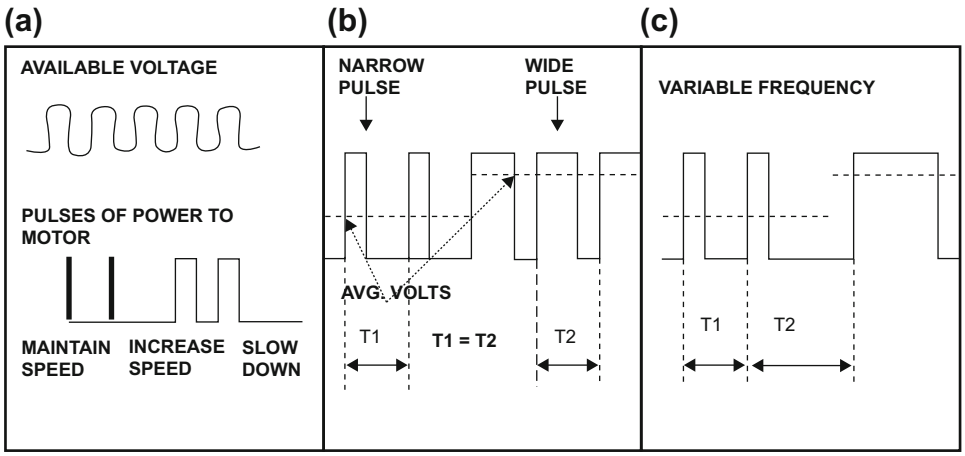


FIGURE 8.18

Servo control types: (a) an SCR control; (b) pulse width determines average voltage; (c) pulse frequency modulation to determine average voltage.

voltage applied to the motor until the right amount of current is produced. Tuning current loops is always complicated work.

One type of semiconductor is the silicon controller rectifier (SCR), which is connected to the AC line voltage (Figure 8.18(a)). This type of device is usually employed where large amounts of power must be regulated, motor inductance is relatively high, and accuracy in speed is not critical (such as constant-speed devices for fans, blowers, conveyor belts). Power out of the SCR, which is available to run the motor, comes in discrete pulses. At low speeds, a continuous stream of narrow pulses is required to maintain speed. If an increase in speed is desired, the SCR must be turned on to apply large pulses of instant power, and when lower speeds are desired, power is removed and a gradual coasting down in speed occurs. A good example of such a system is one car is towing a second car. The driver in the first car is the SCR device and the second car, which is being towed, is the motor/load. As long as the chain is taut, the driver in the first car is in control of the second car. But if the first car slows down slack develops in the chain and, at that point, the first car is no longer in control (and would not be until he gets into a position where the chain is taut again). So, for the periods when the first car must slow down, the driver is not in control. This sequence occurs repeatedly, resulting in a jerky, cogging operation. This type of speed control is, however, quite adequate for many applications.

If smoother speed is desired, an electronic network may be introduced. By inserting a lag network, the response of the control is slowed, so that a large instant power pulse will not suddenly be applied. The filtering action of the lag network gives the motor a sluggish response to a sudden change in load or speed command. This is not important in applications with steady loads or extremely large inertia, but for wide-range, high-performance systems, in which rapid response is important, it becomes extremely desirable to minimize this sluggishness.

Transistors may also be employed to regulate the amount of power applied to a motor. There are several techniques, or design methodologies, used to turn transistors on and off; linear, pulse width modulated (PWM) or pulse frequency modulated (PFM).

Linear mode uses transistors that are activated, or turned on, all the time, supplying the appropriate amount of power required. Transistors act like a water faucet, regulating the appropriate amount of power to drive the motor. If the transistor is turned on halfway, then half of the power goes to the motor. If the transistor is turned on fully, then all of the power goes to the motor and it operates harder and faster. Thus, for the linear type of control, power is delivered constantly, not in discrete pulses (like SCR control), and better speed stability and control is obtained.

Another technique is termed pulse width modulation (PWM), as shown in Figure 8.18(b). In this technique, power is regulated by applying pulses of variable width, that is, by changing or modulating the pulse widths of the power. Compared to SCR control (which applies large pulses of power), the PWM technique applies narrow, discrete (when necessary) power pulses. Operation is as follows: for small pulse width, the average voltage applied onto the motor is low, and the motor's speed is slow. If the width is larger, the average voltage is higher, and therefore motor speed is higher. This technique has an advantage flow power loss in the transistor, that is, the transistor is either fully on or fully off and, therefore, has reduced power dissipation. This approach allows for smaller package sizes.

The final technique used to turn transistors on and off is termed pulse frequency modulation (PFM), as shown in Figure 8.18(c). PFM, regulates power by applying pulses of variable frequency, that is, by changing or modulating the timing of the pulses. For very few pulses, the average voltage applied to the motor is low, and motor speed is slow. With many pulses, the average voltage is increased, and motor speed is higher.

8.3.3 Distributed servo control

There are many ways of putting together servo motion systems that combine hardware and software to meet application requirements. Some controllers operate on multiple platforms and networking buses, with units providing analog output to a conventional amplifier, as well as units that provide current control and direct pulse width modulation (PWM) output for as many as tens or even hundreds of motors simultaneously. There are amplifiers that still require potentiometers to be adjusted for the digital drives' position, speed, and current control. A near-limitless mixing and matching of these units is possible.

Advances in hardware continue to make possible faster and note precise motion control products. New microprocessors provide the tools necessary to create better features, functionality, and performance for lower cost. These features and functionality enhancements, such as increasing the number of axes on a motion controller or adding position controls to a drive, can be traced back to advances in the electronics involved. As dedicated servo and motion control performance has improved, the system-level requirements have increased. Machines that perform servo actions now routinely operate with more advanced software and more complex functions on their host computers. Companies that provide servo or motion systems are differentiating themselves from their competition by the quality of the operating systems they deliver to their customers.

Traditional servo systems often consist of a high-power, front-end computer that communicates to a high-power, microprocessor-based, multiple-axis motion controller card, that in turn interfaces with

a number of drives or amplifiers, which often have their own high-end processors on board. In a number of these cases, the levels of communication between the high-end computer and the motion controller can be broken down into three categories: (1) simple point-to-point moves with noncritical trajectories; (2) moves requiring coordination and blending, with trajectory generation being tied to the operation of the machine; and (3) complex moves with trajectories that are critical to the process or machine.

Increasingly, a significant case can be made for integrating motion control functionality into one compact module to enable mounting close to the motors, providing a distributed system. By centralizing the communications link to the computer from this controller/amplifier, system wiring would be reduced, thereby reducing cost and improving reliability. By increasing the number of power stages, the unit can drive more than one motor independently, up to the processing power of the microprocessor. If size reductions are significant enough, the distributed servo controller/amplifier does not have to be placed inside a control panel, thus potentially reducing system costs even further. Figure 8.19 illustrates such a configuration for distributed servo control system.

To implement such a distributed servo controller and amplifier, the unit must meet the basic system requirements; it should have enough processing power to control all aspects of

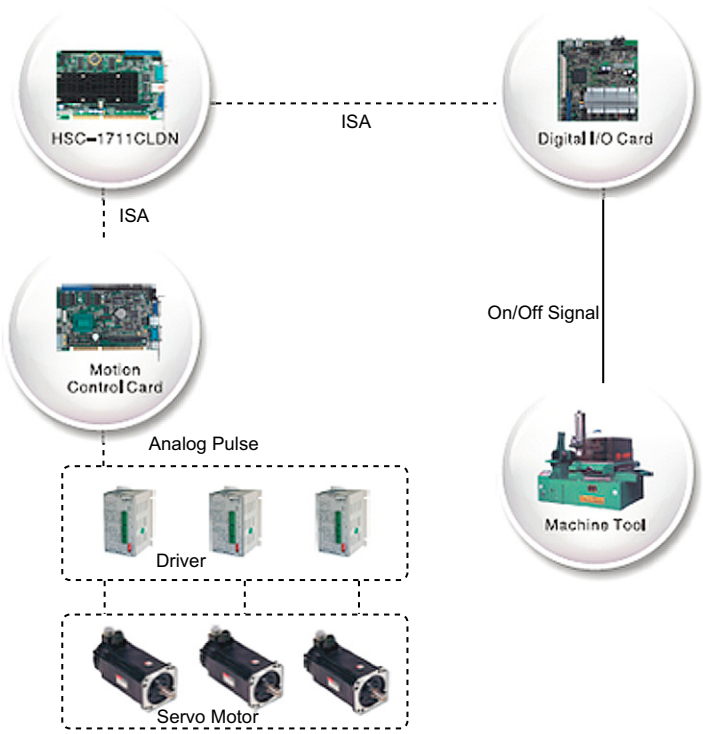


FIGURE 8.19

A schematic distributed servo motion control system.

a multiple-axis move, provide fast enough communications between the computer and the servo controller/amplifier to do the job, and be small enough to satisfy distributed-system requirements, and be reasonably priced.

There are two factors affecting such applications; motion control complexity and network communication requirements.

(1) Motion control complexity

The three basic levels of motion complexity require appropriate features and functions from a distributed servo controller and amplifier. Two types are possible for point-to-point moves with noncritical trajectory applications: those for repetitive point-to-point moves with external-event-generated conditions, and others for point-to-point destinations that vary based on user- or machine-generated events. A typical application for this type of system is a "pick and place" robot. The robot's purpose can vary from moving silicon wafers between trays to placing parts from an assembly line into packing boxes.

A distributed servo controller and amplifier needs to determine a trajectory based on current location and requires either a user or a pre-programmed destination. Effectively, the user need only set acceleration, velocity, jerk parameters (for S-curve), and a series of final points.

Applications that require complex moves with critical trajectories such as inverse kinematics often include the trajectory planning as part of their own intellectual property. The generation of this trajectory is done on a computer platform and then transferred to the motion controller. Typically, significant engineering effort is put into the top-level software provided with the robot. This pre-calculates the trajectory and transfers this information to the controller, with the result that a conventional motion controller will be underused.

Meanwhile, system integration continues to advance on all fronts. All major value-adding components of motion control systems will soon have to comply with the demands for faster controllers with high-speed multiple-axis capabilities supplying commands in multitasking applications.

(2) Network communication requirements

As a unit, there are minimal communication requirements between host and controller that use simple commands and feedback. They include a resident motion program for path planning; support for I/O functionality; support for a terminal interface via such as RS-232; use without resident editor/compiler; and stand-alone motion control. A complete motion control application is programmed into the distributed control module for stand-alone mode control of servo control applications. This mode is typically used for machines that require simple, repetitive sequences operating from such as an RS-232 terminal.

Communication for the stand-alone mode can be simple and is often not time-critical. Simple point-to-point RS-232 communication is often acceptable for applications requiring few motors. Multiple-drop RS-485 or RS-1394 is suitable for applications with many axes of motion and motor networks.

The requirements for the "blended moves with important trajectory" option involve additional communications to a host computer, as the moves involve more information, and speed and detail of feedback to the host are critical. Typical of this type a computerized numerical control system designed to cut accurate, repeatable paths. The trajectory generation and following must be

smooth, as there is a permanent record of the cut. The distributed servo control module must be capable of full coordination of multiple axis control, for which communication to the host is critical. Latency times between issuing of the command and motor reaction can cause inconsistencies in the motion.

In coordinating motion between multiple motors, there are several possibilities. With a single distributed servo controller and amplifier module using multiple amplifiers, coordination can be achieved by the same controller. For applications requiring coordination between motors connected to different controllers, it is possible to achieve this with a suitable choice of high-bandwidth network.

8.3.4 Important servo control devices

This subsection introduces those devices which are important to servo control systems: servo controllers; servo motors; and feedback devices.

(1) Servo controllers

Most servo controllers are in one of three classes: speed servo controllers; position servo controllers; integrated servo controllers. Advanced servo controllers are made with both PIC (programmable integrated circuits) and R/C (radio control) technologies, so an understanding of the digital logic-circuits used is important for an understanding of how the servo controllers work.

First, let us look at a classic position servo controller a three-axis XYZ-table servo controller. The servo control system for this system includes two modules: software and hardware.

The software module is a microprocessor, which has programs handling communication between the control chip and PC, process control of three-axis servo movement, and movement tracking. The hardware module is an FPGA (field-programmable gate array) which includes the

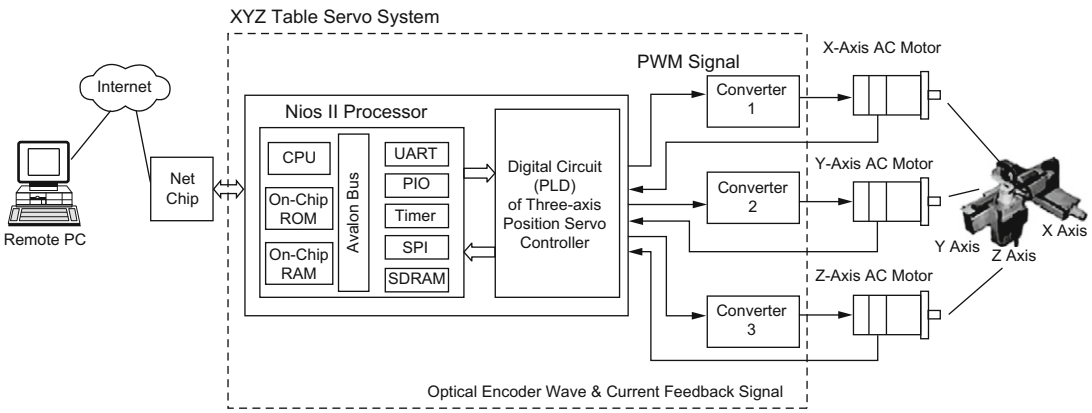


FIGURE 8.20

Block diagram of a three-axis XYZ-table servo control system.

(Courtesy of Southern Taiwan University of Technology, 2008.)

functions that control the position of three motors on the three-axis XYZ-table, a six-group programmable-interrupt controller algorithm computation, a three-group optical encoder signal-detection circuit, a three-group current-estimation circuit, a three-group vector-control coordinate conversion circuit, and a three-group space vector pulse width modulator (PWM) signal output. Figure 8.20 is the block diagram of such a system.

Radiocontrolled (R/C) servos have enjoyed a big comeback in recent years due to their adoption by a new generation of robotics enthusiasts. Driving these versatile servos requires the generation of a potentially large number of stable pulse width modulated (PWM) control signals, which can be a daunting task. A simple solution to this problem is to use a dedicated serial servo controller board, which handles all the details of the multichannel PWM signal generation while being controlled through simple commands issued on a standard serial UART. We will now introduce an array of 32 parallel channels (which may be extended into 64, 128, or more channels) that combines the brute force of a FPGA and the higher-level intelligence of the MCU (microprocessor control unit) to achieve some impressive specifications.

Figure 8.21 gives a block diagram of this type of digital servo controllers' logic circuits. An array of 32 parallel channels of 16-bit accuracy, 12-bit resolution PWM generation units is implemented inside a FPGA. A MCU is used at the heart of the system. Its external memory bus interfaces with the memory-mapped array of 64 PWM registers (i.e., 32 × 16 bits) inside the FPGA. The MCU include initializes the FPGA registers with user-configurable servo startup positions stored in the internal EEPROM. In response to an external interrupt occurring at every PWM cycle, all current servo position values are refreshed in the memory-mapped PWM registers of the FPGA. This is done as a simple memory-to-memory transfer (Figures 8.22 and 8.23).

(2) Servo motors

Electric motor design is based on the placement of conductors (wires) in a magnetic field. A winding has many conductors, or turns of wire, and the contribution of each individual turn adds to the intensity

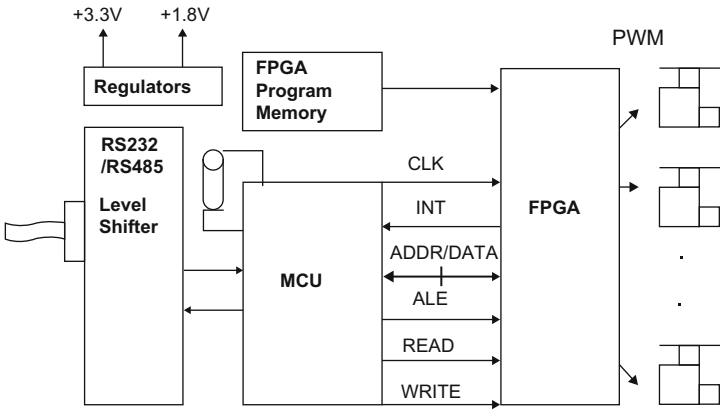


FIGURE 8.21

Overall digital servo controller circuit block diagram.

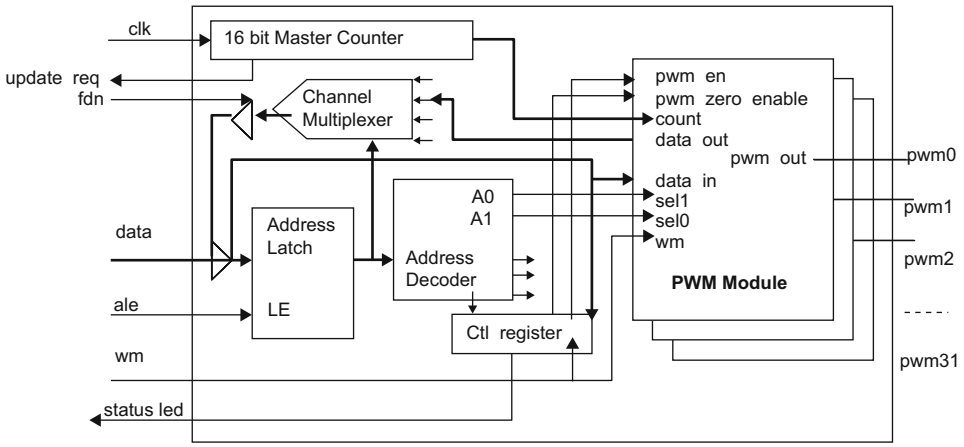


FIGURE 8.22

Internal architecture of FPGA for the digital servo controller circuit given in Figure 8.21.

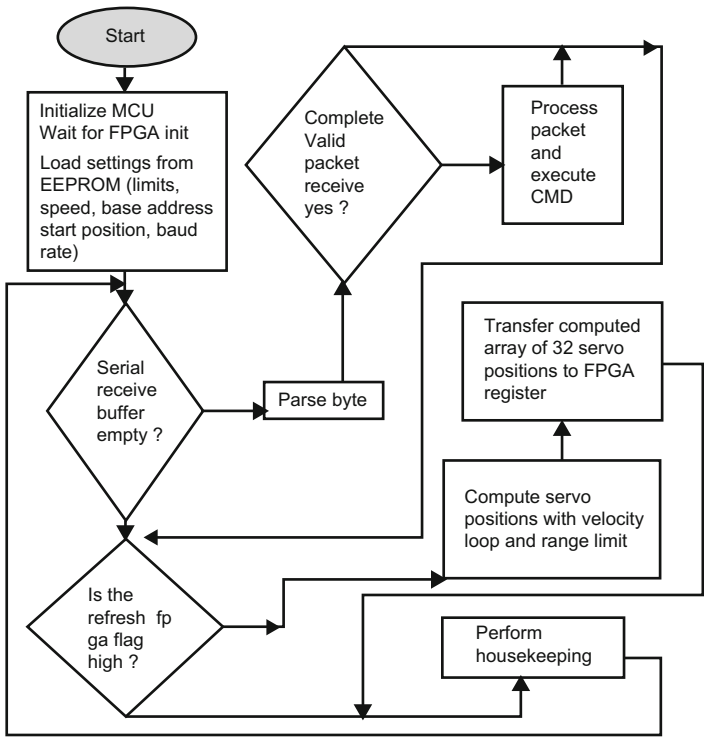


FIGURE 8.23

The MCU firmware flowchart for the digital servo controller circuit given in Figure 8.21.

of the interaction. The force (or torque) developed from a winding is dependent on the current passing through the winding and the magnetic field strength. If more current is passed through the winding, then more force (or torque) is obtained. In effect, the interaction of two magnetic fields causes movement; the magnetic field from the rotor and the magnetic field from the stators attract each other. This is the basis of both AC and DC motor design.

(a) AC motors

AC motors are relatively constant speed devices. The speed of an AC motor is determined by the frequency of the voltage applied (and the number of magnetic poles). There are two types; induction and synchronous.

(1) Induction motor. If the induction motor is viewed as a type of transformer, it becomes easy to understand. Typically, they consist of (1) a stator with laminations and turns of copper wire and (2) a rotor, constructed of steel laminations with large slots on the periphery, stacked together to form a "squirrel cage" rotor. By applying a voltage onto the primary (the stator) of the transformer winding, a current flow results and induces current in the secondary (rotor) winding. One magnetic field is set up in the stator, and a second is induced in the rotor. The interaction of these two magnetic fields results in motion. The speed of the magnetic field around the stator will determine the speed of the rotor. The rotor will try to follow the stator's magnetic field, but will "slip" when a load is attached. Therefore, induction motors always rotate more slowly than the stator's rotating field.

(2) Synchronous motor. The synchronous motor is basically the same as the induction motor but has slightly different rotor construction, which enables this type of motor to rotate at the same speed (in synchronization) as the stator field. There are basically two types of synchronous motors: self-excited (as the induction motor) and directly excited (as with permanent magnets).

The self-excited motor (which can be called reluctance synchronous) includes a rotor with notches, or teeth, on the periphery. The number of notches corresponds to the number of poles of the stator. Often the notches or teeth are termed salient poles. These salient poles create an easy path for the magnetic flux field, thus allowing the rotor to "lock in" and run at the same speed as the rotating field. A directly excited motor (which can be called hysteresis synchronous, or AC permanent magnet synchronous) includes a rotor with a cylinder of a permanent magnet alloy. The magnetic north and south poles are effect, are the salient teeth of this design, and therefore prevent slip.

(b) DC motors

DC motor speeds can be easily varied, so they are used in applications where speed control, servo control, and/or positioning is needed. The stator field is produced by either a field winding or by permanent magnets. This is a stationary field (as opposed to the AC stator field, which rotates). Passing sets up the second field, the rotor field, and current through a commutator and into the rotor assembly. The rotor field rotates in an effort to align itself with the stator field, but at the appropriate time (due to the commutator) the rotor field is switched. In this method then, the rotor field never catches up to the stator field. Rotational speed (i.e., how fast the rotor turns) is dependent on the strength of the rotor field. In other words, the more voltage on the motor, the faster the rotor will turn.

DC motors include shunt wound motors, series wound motors, and compound wound motors as well as such electromechanical actuators as stepper motors and the permanent magnet DC motors.

(3) Feedback devices

Servos use feedback signals for stabilization, speed, and position information, which may be supplied by a variety of devices such as an analog tachometer, a digital tachometer (optical encoder), or a resolver. Each of these devices will now be defined and the basics will be explored.

(a) Analog tachometers

Tachometers resemble miniature motors. In a tachometer, the gauge of wire is quite fine, so the current handling capability is small. It is not used as a power-delivering device; instead, the shaft is turned by some mechanical means and a voltage is developed at the terminals (a motor in reverse!). The faster the shaft is turned, the larger the magnitude of voltage developed (i.e., the amplitude of the signal is directly proportional to speed). The output voltage shows a polarity (+ or −) that is dependent on the direction of rotation.

Analog, or DC, tachometers, as they are often termed, provide directional and rotational information and so can be used to provide speed information to a meter (for visual speed readings) or to provide velocity feedback (for stabilization purposes). The DC provides the simplest, most direct way of doing this.

(b) Digital tachometers

A digital tachometer, often termed an optical encoder or simply an encoder, is a mechanical-to-electrical conversion device. The encoder's shaft is rotated and gives an output signal, which is proportional to distance (i.e., angle) the shaft is rotated through. The output signal may be square, or sinusoidal waves, or provide an absolute position.

Thus, encoders are classified into two basic types: absolute encoders and incremental encoders. The absolute encoder provides a specific address for each shaft position through 360°. This type of encoder employs either contact (brush) or noncontact methods of sensing position. However, the incremental encoder provides either pulses or a sinusoidal output signal as it is rotated through 360°. Thus, distance data are obtained by counting this information.

(c) Resolvers

Resolvers look similar to small motors that is, one end has terminal wires, and the other end has a mounting flange and a shaft extension. Internally, a "signal" winding rotor revolves inside a fixed stator. This represents a type of transformer: when one winding is excited with a signal, through transformer action the second winding is excited. As the first winding is moved (the rotor), the output of the second winding changes (the stator). This change is directly proportional to the angle that the rotor has been moved through.

As a starting point, the simplest resolver unit has a single winding on the rotor and two windings on the stator (located 90° apart). A reference signal is applied to the primary (the rotor), and then, via transformer action, this is coupled to the secondary. The secondary output signal is a sine wave proportional to the angle (the other winding would be a cosine wave), with one electrical cycle of output voltage produced for each 360° of mechanical rotation. These are fed into the controller. Inside the controller, a resolver to digital (R to D) converter analyzes the signal, producing an output representing the angle that the rotor has moved through, and an output proportional to speed (how fast the rotor is moving).

There are various types of resolvers. The type described above would be termed a single-speed resolver; that is, the output signal goes through only one sine wave as the rotor goes through 360 mechanical degrees. If the output signal went through four sine waves as the rotor goes through 360 mechanical degrees, it would be called a four-speed resolver. Another version utilizes three windings on the stator and would be called a synchronizer. The three windings are located 120° apart.

Problems

1. A combined PID controller is able to perform three control mechanisms automatically: proportional control + integral control + derivative control. An illustration of a system using PID control is shown in Figure 8.4. In this system, it needs a precise oil output flow rate. The flow rate is controlled by pump motor speed. The pump motor speed is controlled through a control panel consisting of a variable speed drive. In turn, the drive's speed control output is controlled by an electronic controller. Please analyze this control system and the controller, then explain what proportional control is. What is integral control? What is derivative control?
2. Plot the block diagram of a home central heating system including a boiler; then analyze this system and work out whether it use proportional control, integral control, or derivative control. Or is it a combination of any two or three of these?
3. Please explain why all PID controllers need to be tuned.
4. Trial and error is an easy way of tuning a PID controller, but satisfactory performance is not guaranteed. An example of such a procedure is given in the zone based tuning rule which means that the low and high frequency parts of the controller can be tuned separately, starting with the high frequency part. Please search the Further Reading for an explanation of the trial and error tuning rule, and the zone based tuning rule.
5. In subsection 8.1.4, the PID software uses an interrupt mechanism. However, the event broker paradigm in subsection 16.6 can be used to replace the interrupt mechanism in the PID software. Please plot a flow chart of an event service routine in PID software (assume you are using C++).
6. Please read the original ANSI/ISA 88.01 1995 document to gain an understanding of this batch control standard as far as possible.
7. Please give the number and scale of control hierarchy levels possibly existing in the plant model shown in Figure 8.14.
8. Please translate a typical step of batch process control in Figure 8.11 into the Simatic Batch control center diagram in Figure 8.12(a) and (b).
9. Please read the Simatic Batch and Simatic PCS7 documentation to gain a better understanding of them.
10. Can you find two other high level tools developed by other companies for implementing batch process control apart from Simatic Batch developed by Siemens AG?
11. By an internet search for servo controller products, drive/amplifiers, and feedback devices, please get their specification data.
12. With reference to Figure 8.17; please plot the three types of motion profiles used with servo control systems: constant velocity, trapezoidal, and S curve motion profiles.
13. Please explain why tuning/compensation techniques are necessary for servo controls.
14. To drive an AC motor of constant velocity to rotate a further 90 degrees, please plot the driving pulse diagram (similar to Figure 8.18) using: (1) the PWM technique; (2) the PFM technique.
15. Please identify whether the three axis XYZ table servo system in Figure 8.20 can perform closed loop control.

Further Reading

Dingyu Xue, Yangquan Chen, P. Atherton. Chapter 6: PID controller design. Linear Feedback Control. pp. 183 235. SIAM. 2007.

National Instruments (http://www.ni.com). PID control of continuous processes. http://zone.ni.com/devzone/cda/tut/p/id/6951. Accessed: March 2009.
Lightwave (http://www.ilxlightwave.com). PID control loops in temperature control. http://www.ilxlightwave.com/selection guide/selection guide.html. Accessed: March 2009.
Vance J. VanDoren. Understanding PID control. http://www.controleng.com/archives/2000/ctl0601.00/000601.htm. Accessed: March 2009.
G.M. van der Zalm. Tuning of PID type Controllers: Literature Overview. DAF. 2006.
Embedded.com (http://www.embedded.com). PID controls. http://www.embedded.com/2000/0010/0010feat3.htm. Accessed: March 2009.
The Instrumentation, Systems, and Automation Society (ISA). ANSI/ISA 88.01 1995. 1995.
Wikipedia (http://en.wikipedia.org/wiki). ANSI/ISA 88. http://en.wikipedia.org/w/index.php?title=ANSI/ISA88. Accessed: March 2009.
Wikipedia (http://en.wikipedia.org/wiki). ANSI/ISA 95. http://en.wikipedia.org/w/index.php?title=ANSI/ISA95. Accessed: March 2009.
Hugh Jack. 2001. 6.2 Batch processing. http://www.eod.gvsu.edu/eod/hardware/hardware 38.html. Accessed: March 2009.
Siemens AG (http://w1.siemens.com/entry/cc/en/). SIMATIC BATCH. http://pcs.khe.siemens.com/index simatic batch 6800.htm. Accessed: March 2009.
Ronald E. Menendez, Darrell Tanner. Unified Manufacturing Control System (MCS) Architecture for Pharmaceutical and Biotech Manufacturing. SPE. 2007.
Burkert (http://www.burkert.com). Digital Batch Controllers. 2007.
Prodigy SCADA (http://www.prodigyscada.com). Batch and recipe handling. http://www.prodigyscada.com/index.php/article/view/146. Accessed: March 2009.
Zi ARGUS (www.zi argus.com). Batch Control System. 2008.
Advanced Motion Control (http://www.a m c.com). General servo systems. http://www.a m c.com/content/support/supmain.html. Accessed: March 2009.
Baldor (http://www.baldor.com). Servo control facts. http://www.baldor.com/pdf/manuals/1205 394.pdf. Accessed: March 2009.
George W. Younkin. Industrial Servo Control Systems: Fundamentals and Applications (2nd edition). CRC Press. 2002.
Mclennan (http://www.mclennan.co.uk). Precision motion control. http://www.mclennan.co.uk/technicalmanuals.html. Accessed: March 2009.
Chuck McMains. 2006. PIC based speed controller. http://www.mcmanis.com/chuck/Robotics. Accessed: March 2009.
ELM (http://elm chan.org). Servo Motor Controller: http://elm chan.org/cc e.html. Accessed: March 2009.
Southern Taiwan University of Technology (http://www.stut.edu.tw). SOPC based servo control system for the XYZ table. http://www.altera.com/literature/dc/2.9 2005 Taiwan 3rd SouthernTaiwanU web.pdf. Accessed: March 2009.

CHAPTER

Industrial computers

9

Computers are programmable electronic devices that accept data, execute prerecorded instructions, perform mathematical and logical operations, and output results. In computers, software provides information to a central processing unit (CPU) that executes instructions and controls the operation of other hardware components. Memory allows computers to store data and programs temporarily. Mass storage devices such as tape and disk drives provide long-term storage for large amounts of digital data. Input devices such as a keyboard and mouse allow users to enter information that can be output on a display screen, disk, printer, or personal digital assistant. A computer also requires one or more bus devices to transmit data from location to location, or from component to component.

Computers are generally classified by their hardware and mechanical size, and their processing or computing power. They feature a variety of CPU types and are available with many different operating systems. Spreadsheet, database, word processing, and graphic design applications provide specific functionality. Table 9.1 lists some computer classes and types, including microcomputers, minicomputers, mainframes, and supercomputers.

Of all computer classes, supercomputers are the fastest, most expensive machines, used in applications that perform large numbers of mathematical calculations. The main difference between supercomputers and mainframes is that the former execute a few programs as quickly as possible whereas the latter execute many programs concurrently. Minicomputers are mid-range devices that are smaller and less powerful than mainframes, but larger and more powerful than workstations, client computers that connect to a network and share data with other machines.

Computers can be mounted on panels or racks and often include a monitor for displaying information. Cathode-ray tube (CRT) devices use an electron beam to illuminate phosphor dots and are suitable for applications that require relatively high screen resolutions. Flat panel displays, which are often very thin, are used with portable or laptop computers and include technologies such as liquid crystal display (LCD) and gas plasma.

Industrial computers are those computers made for special industrial applications, which include control and automation, business and production management, logistics and so on. As industrial control, automation and production continue to evolve along side computer-based control and manufacturing, industrial computers become ever more important. In all kinds of industrial control system (embedded, real-time, and distributed control systems) and for all kinds of industrial control engineering (process control, motion control, and production automation), industrial computers and controllers play a key role. In fact, many automated manufacturing processes such as stock control and dispatch are controlled by these computers.

Figure 9.1 shows two photographs of computer-controlled manufacturing systems. The upper panel shows a flexible manufacturing cell in which the host computer can control procedures from machining to measuring. The lower panel shows an integrated manufacturing cell in which several industrial robots are controlled by industrial computers using a simulation-based mechanism.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10009-9

Table 9.1 Computer Classes

Computer Classes	Types	Descriptions
Microcomputers (personal computers)	Desktop computers	Desktop computers are not portable, but are put on a desk or table. They hold the motherboard, drives, power supply, and expansion cards, a separate monitor, keyboard and mouse.
	Laptop (notebook) computers	Laptop or notebook computers are to be carried around with the user. They typically have a built-in LCD screen, keyboard and other built-in pointing device.
	Handheld computers	Handheld computers offer greater portability than laptops. They typically use a touch-sensitive LCD screen for both output and input. They communicate with desktop computers either by cable connection, infrared (IR) beam, or radio waves.
	Palmtop computers	The palmtop computer is a very small microcomputer that typically looks more like a tiny laptop, with a flip-up screen and small keyboard.
	Workstation computers	Workstation computers are powerful, high-end microcomputers. They contain one or more microprocessor CPUs. They may be used for rendering complex graphics, or performing intensive scientific calculations. They may alternately be used as server computers that supply data files to client computers over a computer network.
Minicomputers		A minicomputer is a multi-user computer that is less powerful than a mainframe. This class of computers became available in the 1960s, and has been largely taken over by high-end microcomputer workstations.
Mainframes		A mainframe computer is a large, powerful computer that handles the processing for many users simultaneously (up to several hundred users). Users connect to the mainframe using terminals and submit their tasks for processing by the mainframe. A terminal is a device that has a screen and keyboard for input and output, but it does not do its own processing component.
Supercomputers		A supercomputer is a mainframe computer that has been optimized for speed and processing power. A supercomputer has many more processors than other types. Supercomputers are used for extremely calculation-intensive tasks such as simulating nuclear bomb detonations, aerodynamic flows, and global weather modeling.

FIGURE 9.1

Two photos of real computer-controlled manufacturing systems. (a) The flexible manufacturing cell that contains a CNC vertical lathe, CNC horizontal milling machine, coordinate measuring machine and automatic transportation system can execute multiple-shape and highly accurate product manufacturing. The host computer can control the procedure from machining to measuring. (b) The computer-integrated manufacturing area contains several state-of-the-art CNC machines and an automated storage and retrieval system and several industrial material-handling robots. A simulation-based control system is utilized for computer-integrated control of the work cells.

Industrial computers are different from office or home computers, and also those embedded in non-industrial electronic products, such as game and entertainment machines. These differences are dictated by their functions. Industrial computers are usually designed to perform massive data-processing or other large-scale operations, unlike office or home computers, which are designed small tasks and fewer data. In most cases, industrial computers have high processing speeds, bigger storage volumes, larger display screens, and multifunctional human machine interfaces.

Because they are used in industrial environments, industrial computers are specifically designed to withstand various environmental factors such as shock, vibration, humidity, electromagnetic interference, radio frequency interference, dust, flashes and mist. Accordingly, industrial computers require proprietary software and hardware, and mechanical platforms made of special materials to perform well under harsh conditions.

This chapter discusses the classes, configurations, types, peripherals and accessories of industrial computers. It is aimed at providing concise information on these topics, to give a basic understanding and knowledge of the topic.

9.1 INDUSTRIAL COMPUTER CLASSES AND CONFIGURATIONS

Industrial computers are available in several classes and configurations with a variety of CPUs, displays, I/O interfaces and enclosures. They can be industrial motherboards, industrial single-board computers, industrial embedded computers, and industrial personal computers and workstations. Choices for configuration or mounting style include desktop, portable or notebook, panel or rack. Desktop computers are enclosed for desktop applications. Classes, configurations, mounting style, processor type, processing speed, resistance to harsh conditions, memory, storage, and display are the most important parameters to consider when looking for industrial computers. Additional specifications include I/O interfaces, expansion slots, power supplies, features, environmental parameters and so on.

9.1.1 Industrial computer classes

Industrial computers can be industrial motherboards, industrial single-board computers, industrial embedded computers, industrial personal computers, and workstation computers.

(1) Industrial motherboards

Motherboards are the foundation of all computers. They are printed-circuit boards (housing the computer's basic circuitry and vital components), into which other boards, or cards, are plugged. They manage all data transactions between the CPU (central processing unit) and the peripheral devices. Their main components are memory, CPU, microprocessor and coprocessors, BIOS, interconnecting circuitry, controllers, mass storage, parallel and serial ports, and expansion slots. Other component devices and peripherals can be added through expansion slots, including the monitor screen, the keyboard, disk drives, etc.

There are two important differences between industrial and standard motherboards. Firstly, industrial motherboards need to be more rugged than standard models. Since they are used in tougher conditions than standard computer motherboards, industrial motherboards, like industrial LCD displays or industrial LCD touch screens, need to be able to cope with higher temperatures and need to be tougher in case of a shock or a hit. Secondly they need to be able to easily handle the massive amounts of information which are essential to industrial operations and controls. They are used also for many critical functions and applications, such as industrial control, the military, aerospace industries, and the medical field.

The interface between computer motherboards and smaller boards or cards in the expansion slots is called the bus. Industrial motherboards communicate with and control peripheral devices via buses or other communication standards. Some of the most common buses are ISA (Industrial Standard Architecture), ESA (Extended Industrial Standard Architecture), and PCI (Peripheral Component Interconnect).

As an illustration, Figure 9.2 is a photograph of the ATX-945G Industrial ATX motherboard by Quanmax Corporation; Figure 9.3 is the system block diagram of the ATX-945G Industrial ATX motherboard. Table 9.2 gives its product specifications.

(2) Industrial single-board computers

A single-board computer has one circuit board that contains all the components (i.e. processor, memory, I/O, drive, bus, interface, etc.) necessary to function as a complete digital computer. In other words, single-board computers are those, and which are completely built on a single circuit board are composed of a microprocessor, memory chip, and serial and parallel interfaces to communicate with other devices. Processor or CPU type, processor speed, I/O bus specifications, memory, I/O interfaces, and storage types and volumes are important specifications to consider when looking for single-board computers, along with chipset type, features and environmental parameters.

Because of the very high levels of integration, reduced component counts and reduced connector counts, single-board computers are often smaller, lighter, more power-efficient and more reliable than comparable multiple-board computers. They are found in many applications we use, such as laptops

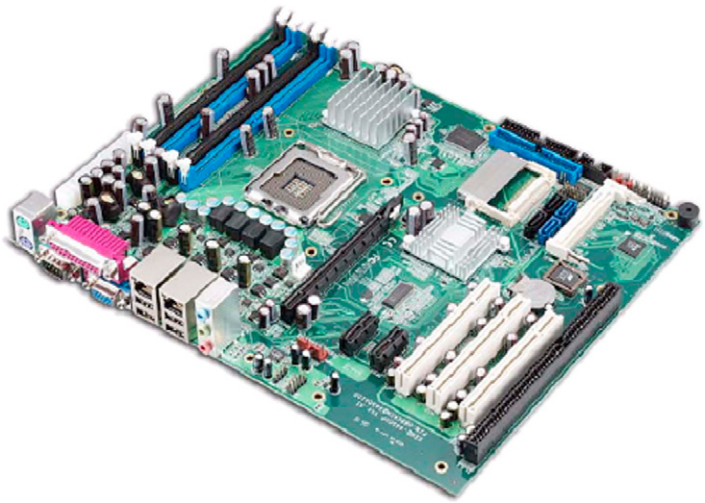


FIGURE 9.2

Photograph of an industrial motherboard ATX-945G.

(Courtesy of Quanmax (www.quanmax.com), 2009.)

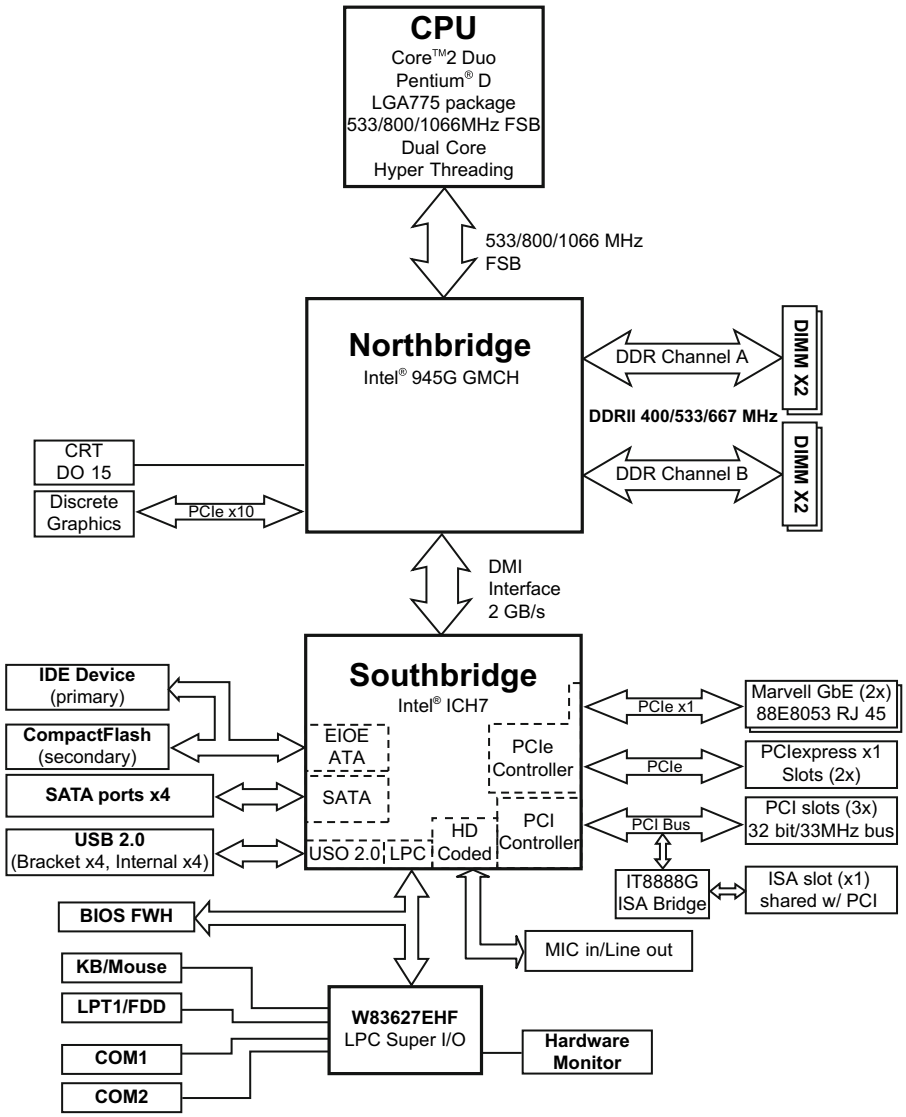


FIGURE 9.3

Industrial motherboard ATX-945G system block diagram.

(Courtesy of Quanmax (www.quanmax.com), 2009.)

and notebooks, or serve as the motherboard for instrumentation systems. Single-board computers are most commonly used in industrial situations. They are designed to be integrated with other electronic components to form an industrial machine for process control; or embedded within other devices to provide control and interfacing.

Table 9.2 Product Specifications of ATX-945G Industrial Motherboard

Form factor	• ATX form factor single-processor industrial motherboard
Processors	• Single processor with Intel® Land Grid Array 775 (LGA775) package • Intel® Core™2 Duo, Pentium® D, Pentium® 4, or Celeron® D Processor • Intel Hyper-Threading Technology supported
Chipsets	• Intel® 945G graphic memory controller hub (GMCH) • 1066/800/533 MHz front side bus • Intel® I/O controller hub 7 (ICH7)
BIOS	• Support for 4 Mb firmware hub • BIOS write protection provides anti-virus capability • DMI BIOS support • Supports system soft power-off (via ATX power supply) • Supports PXE Remote Boot • Supports boot from USB device
System memory	• Dual-channel DDR2 DIMM 667/533/400 MHz, up to 4 DIMMs • Maximum 4GB system memory
Graphics	• GMCH analog RGB output on the faceplate • Resolution up to 2048 × 1536 at 75 Hz. • PCI Express x16 extension slot
PCI/PCIe/ISA support	• 32-bit/33 MHz PCI Bus, 16-bit/8 MHz ISA bus • 1× PCIe x16 slot, 2× PCIe x1 slots, 3× PCI slots, 1× ISA slot (shared w/ PCI)
Clock generator	• Integrated Circuit Systems ICS954101 (CK410)
I/O controller	• LPC Super IO controller W83627EHF
On-board LAN	• Onboard Marvell 88E8053 supports 2 GbE ports on the bracket • Support IEEE 802.3x compliant flow control • Support IEEE 802.3ab Auto-Negotiation of speed, duplex and flow control
Peripheral support	• One IDE channel supports two Ultra ATA 100/66/33 devices • 4 channel SATA-300 • CompactFlash Type II socket • Mini PCI socket • 8×ports USB 2.0 (four on board Type-A, four external) • Two 16550 UART compatible serial ports with ESD protection (one port supports RS232/422/485 interface, jumper-less by BIOS selection) • One internal bi-direction parallel port with SPP/ECP/EPP mode • One internal floppy port • PS/2 keyboard and mouse port • HD audio codec, line-out, MIC-in
Watchdog timer	• 1-255 step (sec/min), can be set with software on Super I/O
Miscellaneous	• Voltage, fans and CPU/system temperature monitoring. • Fan speed control and monitoring
Operating temperature	• 0°C–50°C

Single-board computers are now commonly defined across two distinct architectures: no slots and slot support. The term single-board computer now generally applies to an architecture where it is plugged into a backplane to provide for I/O cards. The most common variety in use is similar to other full-size plug-in cards and is intended to be used in a backplane. In Intel personal computers, the

intelligence and interface/control circuitry is placed on a plug-in board that is then inserted into a passive or active backplane. The end result is similar to having a system built with a motherboard, except that the backplane determines slot configuration.

Embedded single-board computers are boards providing all the required I/O with no provision for plug-in cards. Applications are typically gaming (slot machines, video poker) and machine control. They are much smaller than motherboards, and provide an I/O mix which is more targeted to an industrial application such as on-board digital and analog I/O, on-board bootable flash so no hard drive is required, no on-board video, etc.

The development of personal computers brought a sharp shift away from single-board computers, with construction comprising a motherboard, with functions such as serial ports, disk drive controller and graphics provided on daughterboards. The recent availability of advanced chip sets providing most I/O features as embedded components allows motherboard manufacturers to offer motherboards I/O that also have the traditionally provided by daughterboards. Most that was now offer on-board support for disk drives, graphics, Ethernet, and traditional I/O such as serial and parallel ports, USB (universal serial bus), and keyboard/mouse support. Plug-in cards are now more commonly used for high-performance graphics cards (really graphic co-processors), and specialized uses such as data acquisition and DSP (digital signal processor) boards.

(3) Industrial embedded computers

Embedded computers are all-in-one devices used to handle special-purpose computing tasks for business and industrial applications. Industrial embedded computers, or embedded industrial computers, are those that are incorporated into some system or device that do not inherently require these computers for basic operations. Thus embedded computers are mostly used to improve and expand production processes or operation controls.

Consider the changes that have taken place in automotive electronics: not only are embedded computers being incorporated into very visible devices such as navigation systems, they are also being used for fundamental functions of cars in devices such as braking systems. All devices, from phones to cars, that contain embedded computers create opportunities for the supply of both hardware and software. Although most is specialized for embedded use, much is also common to both computers and embedded computers. This means that demand for embedded computers of various types has further repercussions on demand for everything from semiconductors to software to services.

From this point of view, most industrial computers are assumed to be embedded computers, since they are incorporated into larger industrial systems or devices to fulfill their assigned functionalities. The following are the main types of embedded industrial computer used in industrial controls, production and management.

(a) Embedded single-board computers

Embedded single-board computers are highly integrated embedded computers that build immediate trust by guaranteeing standard form factors that speed development times, increase flexibility, provide for future expansion, and offer scalable performance and advanced features to fill a wide variety of applications. With a strong product strategy, excellent product development and management, flexible global manufacturing capability and leading technology, reliability is built into every embedded single-board computer!

(b) Vertical purpose embedded computers

Some industrial computer manufacturers offer a series of durable and application-ready embedded computer boxes for a range of applications, such as automation, control, transportation, entertainment, surveillance, servers etc. Vertical purpose embedded computers are all-in-one embedded systems with strong mechanical design and excellent anti-shock and anti-vibration properties and thus can be extremely reliable in a harsh environment.

(c) Ruggedized embedded system with expansions

To fulfill customers' diverse needs, there exist a series of outstanding expandable embedded computer systems featuring various types of expansion slots, such as AGP, PCI, PCIexpress, ISA, ESA etc. The user can easily plug in graphic cards, motion cards, or communication cards for system expansion and system integration.

(d) Cost-effective embedded system

Some industrial computer manufacturers provide high-performance yet cost-effective solutions to get to market easily and quickly. The cost-effective embedded systems furnish customers with all their needs such as powerful computing capability, fanless operation, low power use, reliability, flexible I/O configuration, and long product life support.

(e) Industrial barebones system

A full range of rack-mount systems with industrial motherboards such as the ATX / Micro ATX motherboards allow companies to setup barebones applications. Industrial rack-mount barebones deliver powerful computing performance, expansibility, high storage capacity, reliability and cost-effectiveness, and feature solid designs. Also available as complete certified systems, this industrial computer series provides the most popular interfaces, such as Ethernet ports, USB ports, PCI expansion slots, PS/2 ports, and COM ports.

(4) Industrial personal computers and workstation computers

Industrial personal computers, like other kinds of industrial computers, are rugged industrial-grade, models for use in harsh environments. They are available in many types and configurations with a variety of CPUs, displays, memory, I/O interfaces and various mountings. Their most important parameters are: configuration or mounting style, processor type, memory, storage, and display. Additional specifications include I/O interfaces, slots, expansion cards, power supply, features, and environmental parameters.

Industrial workstation computers, like their office counterparts, are used as client/server computers that connect to an industrial network to share data with other machines. They are used in applications such as computer-controlled manufacturing, computer-aided automation, database data entry, graph displaying, and product imaging. In terms of computing power, low-end workstations are equivalent to high-end personal computers while high-end workstations are equivalent to minicomputers.

Computer workstations vary widely in terms of slots and expansion bays. Some use industry standard architecture (ISA) while others use peripheral component interconnect (PCI) technology. Input/output (I/O) ports can be serial, parallel, wireless, or networked. Universal serial bus (USB) is

the standard serial bus for low- to medium-speed peripheral devices. Network protocols include Ethernet and Token Ring. Some computer workstations incorporate wireless local area network (WLAN) technology.

Computer workstations are available with a variety of operating systems and processors. They provide random access memory (RAM) with separate internal and external caches and may even include Flash memory, a type of RAM that can retain information when the computer is powered off.

Computer workstations feature a variety of display, resolution, and user interface options. Cathode-ray tube devices use an electron beam to illuminate phosphor dots and are suitable for users who require relatively high screen resolutions. Flat panel displays, which are often very thin, are used with portable or laptop computers and include technologies such as liquid crystal display and gas plasma. Computer workstations are also available with color or backlit displays, membrane or tactile keyboard interfaces, and voice or handwriting recognition capabilities.

Both industrial personal computers and industrial workstation computers are mostly rack, panel, or wall mounted. These configurations are introduced in the next subsection.

9.1.2 Industrial computer configurations

Computers are configured by equipping them with different computer chassis, mounts, monitors, etc.

FIGURE 9.4

Industrial rack-mount computers: (a) photographs of two rack-mount personal computers; (b) photographs of two rack-mount workstation computers.

(1) Industrial rack-mount computers

Rack-mounted computers are widely used by manufacturing facilities and laboratories. This configuration provides reliability, ease of maintenance and stability. They featured a variety of processor and bus configurations and many supports such as PCI and ISA busses, side-access serial ports, parallel ports, USB port, external monitor port, I/O link, and an Ethernet port, removable hard drive and power supply, along with front-accessible USB and so on. In addition, rack-mounted computers support various embedded operating systems including Windows and embedded Linux. Figure 9.4 shows two groups of photographs for both rack-mounted personal computers and workstations.

(2) Industrial panel-mount computers

Industrial panel computers are a space-saving solution for areas where multiple operators use a single computer. This configuration is cost-effective, highly reliable, and a powerful performance product designed to operate in harsh environments. They featured a variety of processor and bus configurations and many supports such as PCI and ISA busses, side-access serial ports, parallel ports, USB port, external monitor port, I/O link, and an Ethernet port, removable hard drive and power supply, along with front-accessible USB and so on. In addition, panel-mount computers support various embedded operating systems including Windows and embedded Linux. Figure 9.5 shows two groups of photographs for both panel-mount personal computers and panel-mount workstation computers.

FIGURE 9.5

Industrial panel-mount computers: (a) photographs of two panel-mount personal computers; (b) photographs of two panel-mount workstation computers.

(3) Industrial wall-mount computers

There are a number of reasons a computer screen may be required to be fixed to the wall. It could be a space-saving effort, or it could be required for such information as promotional material and company data to be screened within an office or shop environment. A specific type of wall mount is needed depending on the size of the computer display, and the height at which it is to be mounted.

There are two models of wall-mount computers: a basic but more customizable wall-mountable computer station, and a motorized unit that offers less customization. Wall-mount computers attach to a wall via plates that are bolted directly into a secure area of the wall, making an extremely strong attachment. Figure 9.6 shows two groups of photographs for both wall-mount personal and workstation computers.

9.2 INDUSTRIAL COMPUTER PERIPHERALS AND ACCESSORIES

The peripherals and accessories for industrial computers include specialized or proprietary products that are auxiliary to computer mainframes. A common type is the back-up storage or disk drive. Others

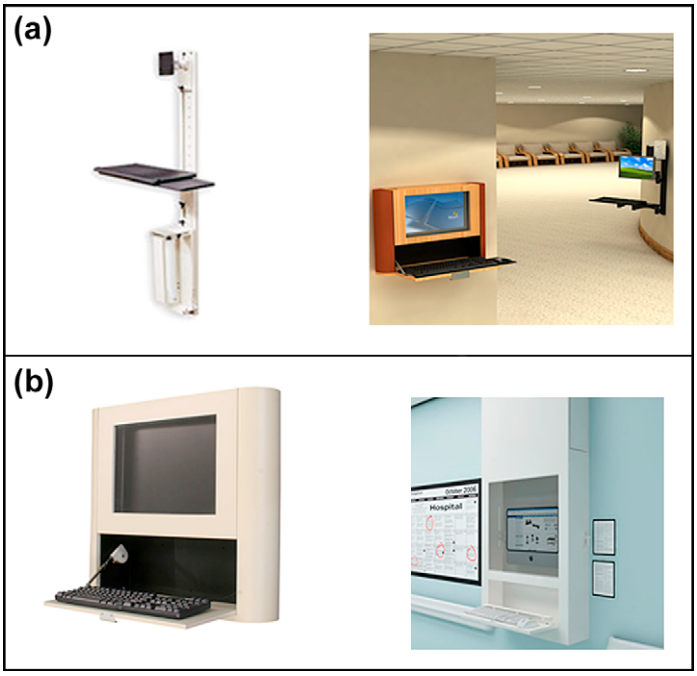


FIGURE 9.6

Industrial wall-mount computers: (a) photographs of two wall-mount personal computers; (b) photographs of two wall-mount workstation computers.

are input devices such as the mouse or other pointers, and output devices such as monitors, screens, and printers. A specialty computer peripheral is attached to an existing computer mainframe through serial or USB ports. A wireless computer accessory is networked to the system through a wireless access point.

Specialty computer peripherals are often industry-specific; for example, graphic designers use special drawing tablets and scanners that are designed for refined illustration or fine art work. Others include laboratory or research equipment that can be networked for data gathering and analysis such as a microscope, spectrometer, or other measurement device.

Specialty computer peripherals for use in industrial environments often need to be more robust and durable than those for an office or home computer. For example, industrial keyboards are built to resist dust and dirt, and some may even be waterproof and illuminated. Industrial computer monitors and displays are also built to withstand harsh environments and include flat panel displays that come with touch screens and stainless steel front bezels.

Tables 9.3 and 9.4 list some typical computer peripherals and accessories, respectively.

Table 9.3 Some Important Industrial Computer Peripherals

Computer Peripherals	Types	Functions
Computer data storage devices	Primary storage	Primary storage includes various memories representing storage devices which are directly accessible to the CPU.
	Secondary storage	Secondary storage includes all types of hard disks used as storage, which are different from primary storage in that they are not directly accessible by the CPU.
	Tertiary storage	Tertiary storage includes various magnetic tapes and optical jukeboxes providing a third level of storage for databases, data deposits and mount/dismount media, etc.
	Offline storage	Offline storage includes floppy disks, CD, DVD, Zip-disks or punched cards used to transfer information as disconnected storages.
Computer network cards	Wired network cards	A network interface card, or network adapter, is typically a small unit of hardware designed to allow the computer to communicate over a computer network with wired connections. Several types of hardware adapters exist: the PCI adapter or PC card, USB adapter, media adapter, Ethernet adapter, etc.
	Wireless network cards	A wireless network interface card is a network card which connects to a radio-based computer network. Several types of wireless network card exist: the PCI adapter or PC card, USB adapter, media adapter, Ethernet adapter, etc.

(*Continued*)

Table 9.3 Some Important Industrial Computer Peripherals *Continued*

Computer Peripherals	Types	Functions
Computer power supplies	Converters and batteries	Computer power supplies convert alternating current (AC) to direct current (DC), which is needed by the computer. Choices for computer power supplies include ATX and others (AT, LPX, NLX, SFX, and TFX).

Table 9.4 Some Important Industrial Computer Accessories

Computer Accessories	Types	Functions
Computer display and output devices	Visual displays and outputs	Visual display devices typically are monitor screens and printers which display data and images generated from the video and paper output equipment of computers.
	Tactile displays and outputs	Tactile display systems include somatosensory devices such as mechanoreceptors for tactile sensation and nociceptors for pain sensation. The sensory information (touch, pain, temperature, etc.) is then conveyed to the central nervous system by afferent neurons.
	Auditory displays and outputs	Auditory display and output devices include earphone, microphone, MP3 cards, etc. which are capable of being heard by humans are called audio or sonic. The range of the sonic frequency is typically considered to be between 20 Hz and 20,000 Hz.
Computer keyboards, mouse and data I/O devices	Standard computer keyboards	In normal usage, the keyboard is used to type text and numbers into a word processor, text editor or other program. Standard keyboards are normally a mechanical or electronic input device, including traditional 104-key Windows keyboards, gaming and multimedia keyboards with extra keys, etc.
	Special computer keyboards	In special usage, keyboards can be chorded, virtual, touch-screen, laser and infrared input devices, and so on.
	Mouse/mice	The computer mouse is a pointer that functions by detecting two-dimensional motion relative to its supporting surface.
	Disk reader/writer	Disk or tape readers/writers are data I/O devices that typically are CD, DVD, magnetic reader/writer devices, etc.
Computer chassis	Rack mount; panel mount; wall mount	The computer chassis, or computer case or enclosure or cabinet, is the enclosure that contains the main hardware and mechanical components of a computer. Computer chassis can be made of steel, aluminum, or plastic, etc.

Problems

1. By searching the Internet, please find examples of microcomputers, mainframes, and supercomputers.
2. With reference to these types of industrial computers: industrial motherboards, industrial single board computers, industrial embedded computers, and industrial PCs and workstations; (1) if using the right PLC programming software, which type of industrial computer is more suitable as a PLC controller? (2) If using the right CNC part programming software, which type of industrial computer is more suitable as a CNC controller? (3) If using the right fuzzy logic programming software, which type of industrial computer is more suitable as a FLC controller?
3. With reference to these types of industrial computers: industrial motherboards, industrial single board computers, industrial embedded computers, and industrial PCs and workstations; (1) if using the right PID programming software, which type of industrial computersis more suitable as a PID controller? (2) If using the right batch control programming software, which type of industrial computer is more suitable as a batch controller? (3) If using the right servo control programming software, which type of industrial computer is more suitable as a SMC controller?
4. Which of these types of industrial computers: industrial motherboards, industrial single board computers, industrial embedded computers, and industrial PCs and workstations, are more suitable to be used in distributed control systems?
5. Which of these types of industrial computers: industrial motherboards, industrial single board computers, industrial embedded computers, and industrial PCs and workstations, could function as computers or controllers in most industrial SCADA systems?
6. Why do the most computers need peripheral and accessory devices?

Further Reading

Chung shan Institute of Science and Technology (http://cs.mnd.gov.tw). System manufacturing. http://cs.mnd.gov.tw/english/Publish.aspx?cnid=1222&Level=1. Accessed: April 2009.

Thomas E. Beach. Computer concepts and terminology. http://www.unm.edu/~tbeach/terms/index.html. Accessed: April 2009.

Quanmax (http://www. Quanmax.com). ATX 945G Industrial ATX Motherboard User's Guide. May 2008.

GlobalSpec (http://www.globalspec.com). Industrial computers and embedded systems. http://industrial computers.globalspec.com/ProductFinder/Industrial Computers Boards. Accessed: April 2009.

GlobalSpec (http://www.globalspec.com). Embedded computers. http://search.globalspec.com/ProductFinder/FindProducts?query=embedded%20computers. Accessed: April 2009.

GlobalSpec (http://www.globalspec.com). Special computer peripherals. http://search.globalspec.com/ProductFinder/FindProducts?query=Specialty%20Computer%20Peripherals. Accessed: April 2009.

Ezine article (http://www.ezinearticle.com). Computers and technology. http://ezinearticles.com/?cat=Computers and Technology. Accessed: April 2009.

Advantech (http://www.advantech.com). Embedded and industrial computing. http://www.advantech.com/products/. Accessed: April 2009.

CHAPTER

Industrial control networks 10

Most modern industrial control systems are embedded networks, relying on built-in, special-purpose, digital or numerical controllers and computers to perform real-time and distributed control applications. They are used by controllers and computers to communicate with each other over dispersed locations, and to interact with large numbers of sensors and actuators.

Embedded networks are common in aircraft, factories, chemical processing plants, and also in machines such as cars. The types of architecture most commonly used are controller area, supervisory control and data acquisition, industrial Ethernet, DeviceNet, HART field, and industrial local area networks.

Their use raises many new practical and theoretical questions about network architecture, components, protocols, compatibility of operating systems, and ways to maximize the effectiveness of the embedded hardware. This part of the textbook provides students and engineers with a broad, comprehensive source of information and technological detail to enable them to address many questions and aspects of real-time and distributed control through use of embedded networks.

10.1 CONTROLLER AREA NETWORK (CAN)

10.1.1 Introduction

The controller area network (CAN) is a highly integrated system using serial bus and communications protocol to connect intelligent devices for real-time control applications. CAN is able to operate at data rates of up to 1 megabits per second over a possible line length of up to several kilometres, and has excellent error detection and confinement capabilities.

CAN was initially created by the German automotive system supplier Robert Bosch in 1985 as a working method for enabling robust serial communication. Since its inception, CAN has gained widespread popularity in industrial control and automation. In 1993, it became the international standard known as International Standard Organization (ISO) 11898. Since 1994, several higher-level protocols have been standardized on CAN, such as CANopen and DeviceNet. Thereafter, CAN has been documented in ISO 11898 (for high-speed applications) and ISO 11519 (for lower-speed applications).

CAN provides an inexpensive, durable network that helps multiple field devices communicate with one another. This allows electronic control units (ECU) to have a single CAN interface rather than separate analog and digital inputs for every device in the system. CAN was first created for automotive use, so its most common application is in vehicle electronic networking. However, as other industries have realized the advantages of CAN , it has been adopted for a wide variety of applications. Railway applications such as streetcars, trams, undergrounds, light railways, and long-distance trains incorporate CAN on different levels of the multiple networks within these vehicles. For example, CAN links the door

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10010-5

units or brake controllers, passenger counting units, etc. CAN has also gained applications in aircraft for flight-state sensors, navigation systems, and research PCs in the cockpit. There are CAN buses in many aerospace applications, ranging from in-flight data analysis to aircraft engine control systems. Medical equipment uses CAN as an embedded network. In fact, hospital operating room components such as lights, tables, cameras, X-ray machines, and patient beds all have CAN-based systems. Lifts and escalators use embedded CAN networks, and hospitals use the CANopen protocol to link lifting devices, such as panels, controllers, doors, and light barriers, to each other and to control them. CANopen is also used in non-industrial applications such as laboratory equipment, sports cameras, telescopes, automatic doors, and even coffee machines.

10.1.2 CAN systems

Because it functions as a network and an interface, CAN has a bus with lines topology. The use of topology components is particularly advantageous in building automation systems to adapt the CAN bus flexibly to the requirements of line routing. Depending on the local network circumstances, CAN repeaters, CAN bridges, or gateways can be used.

CAN networks integrate remote sensors or actuators with stub lines which can be implemented as CAN repeaters. As indicated in Figure 10.1(A), a network is physically separated through the use of CAN repeaters. The achievable stub lines length is only limited by the CAN baud rate used: this is only limited by the length between the two most distant devices in the complete CAN network. It is thus also possible to set up CAN systems in tree or star structures.

In Figure 10.1(B), CAN bridges divide a system into independent networks, so allowing networks with different baud rates to be connected to each other. Using an integrated filter function, messages can be filtered before transmission from one network to another in order to minimize the bus load on the individual networks. In contrast to CAN repeaters, CAN bridges allow the network

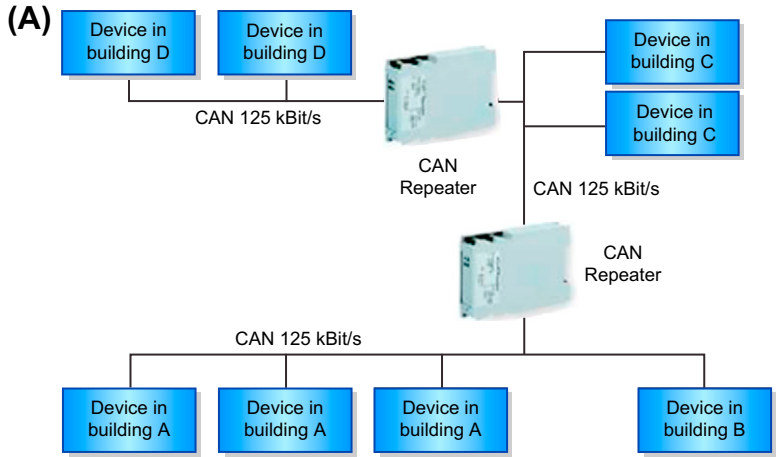


FIGURE 10.1

CAN System with topology components. (A) Cascaded CAN network using CAN repeaters; (B) extended CAN network using CAN bridges; (C) an industrial CAN network using Ethernet as "backbone".

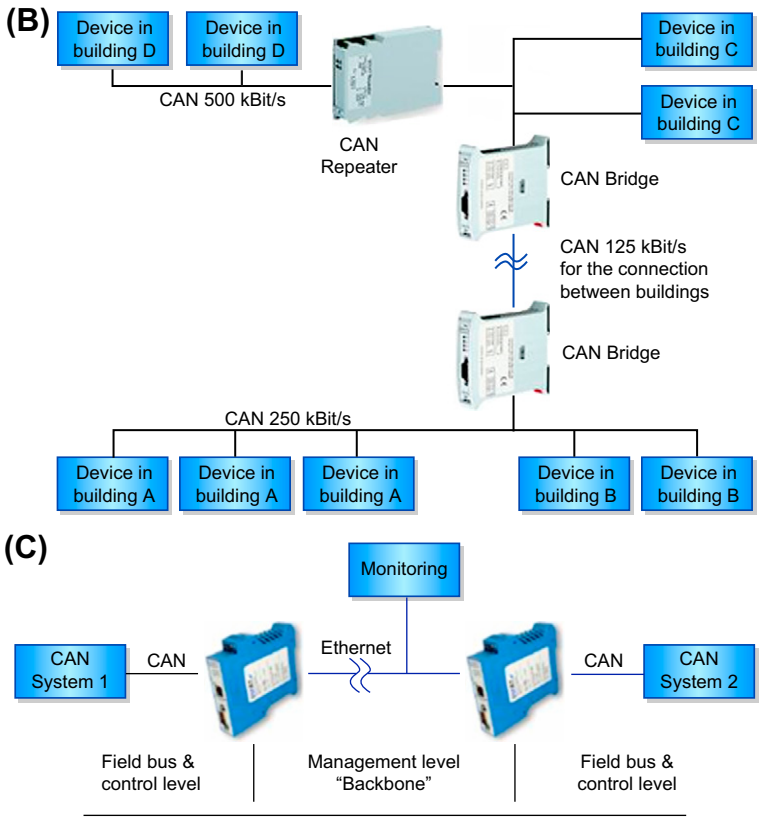


FIGURE 10.1 (*continued*)

to be extended; by dividing it into several sub networks to increase the extension or the bus speed of the overall system: the effective line length of a CAN system can be increased by separating and interposing CAN bridges.

As shown in Figure 10.1(C), gateways are used to connect CAN networks to the same or other network types, for instance to connect management and service computers to CAN networks. Local CAN networks can also be connected via existing WAN/LAN (wide area network/local area network) connections with high transmission rates even with large extensions. It is thus possible to connect CAN networks via Ethernet over large distances with high data transmission rates.

In conclusion, by using topology components, CAN is ideally adapted to the requirements of line routing. Unnecessary cable looms are also avoided, planning, installation and operating costs are reduced, influences on the network due to faults from the outside are minimized and thus greater security is achieved.

Fieldbus networks from the Open System Interconnection (OSI) network model point of view usually only implement layers 1, the physical layer, layer 2, the data-link layer, and layer 7, the

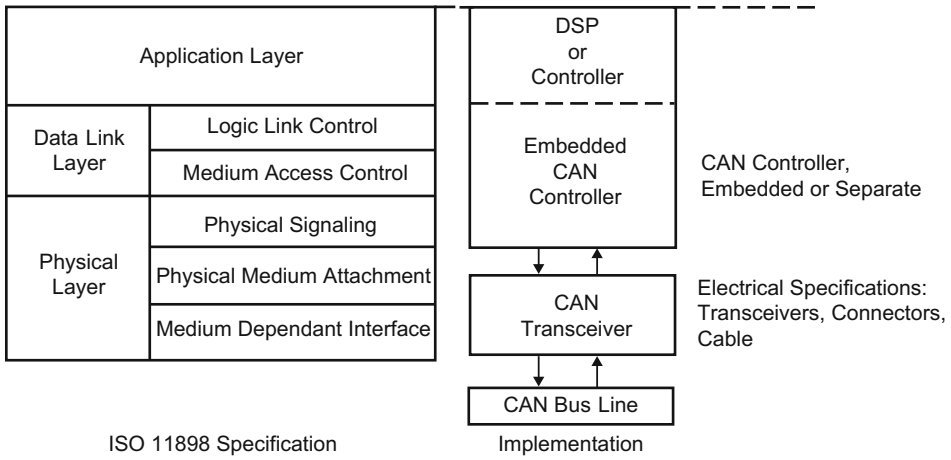


FIGURE 10.2

The layered ISO 11898 Standard Architecture applied for CAN networks.

application layer. The intermediate layers are not needed because a Fieldbus network usually consists of a single network segment only (no need for transport and network layer, and has no notion of sessions or a need for different internal data presentation). Figure 10.2 displays the ISO 11898 standard architecture for CAN networks.

In the ISO 11898 standard architecture, there are three layers from top down: the application layer, the data-link layer, and the physical layer.

(1) Application layer

The application layer provides the high-level communication functions of the OSI layered model. These functions may be implemented by a system software developer or handled by a higher-layer protocol such as:

(a) CAL (CAN application layer)

CAL is based on an existing and proven protocol originally developed by Philips Medical Systems. CAL is an application-independent application layer that has been specified and is now maintained by the CAN in Automation (CiA) user group. CAL provides four application layer service elements:

1. CMS (CAN-based message specification): it offers objects of type variable, event and domain to design and specify how the functionality of a device (a node) can be accessed through its CAN interface exceeding the 8 bytes maximum data content of a CAN message, including an abort transfer feature. CMS derives from MMS (Manufacturing Message Specification), which is an application layer protocol designed for the remote control and monitoring of industrial devices.
2. NMT (network management): it offers services to support network management, e.g. to initialize, start or stop nodes, detect node failures; this service is implemented according to a master slave concept (there is one NMT master).

3. DBT (distributor): it offers a dynamic distribution of CAN identifiers (officially called Communication Object Identifier) to the nodes on the network; this service is implemented according to a master slave concept (there is one DBT master).
4. LMT (layer management): it offers the ability to change layer parameters to change the NMT address of a node, or change the bit-timing and baud-rate of the CAN interface.

(b) CANopen

CAL provides all network management services and message protocols, but it does not define the contents of the CMS objects or the kind of objects being communicated. CANopen was developed to solve this problem. CANopen is built on top of CAL, using a subset of CAL services and communication protocols; it provides an implementation of a distributed control system using the services and protocols of CAL. It does this in such a way that nodes can range from simple to complex in their functionality without compromising interoperability between the nodes in the network.

The central concept in CANopen is the device object dictionary (OD), a concept also used in other Fieldbus systems (Profibus, Interbus-S). Every node in the network is assigned an OD containing all parameters that describe the device and its network behavior. The CANopen object dictionary (OD) is an ordered grouping of objects; each object is addressed using a 16-bit index. To allow individual elements within data structures to be accessed, an 8-bit sub-index has been defined as well. The CANopen device model with OD is displayed in Figure 10.3 and the general layout of the CANopen OD is shown in Table 10.1.

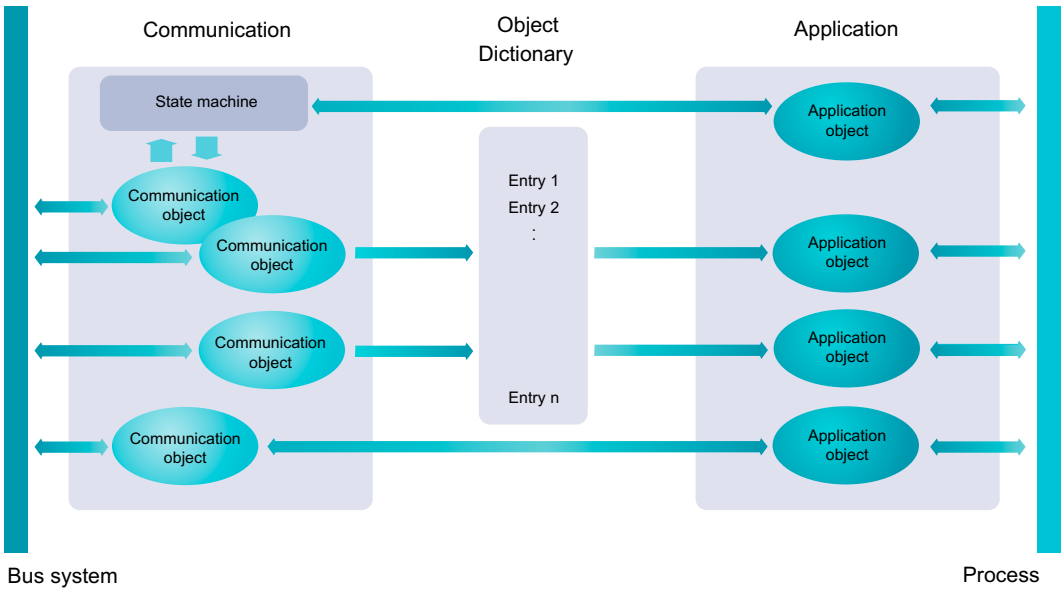


FIGURE 10.3

CANopen device model.

Table 10.1 General CANopen Object Dictionary Structure

Index	Object
0000	Not used
0001 – 001F	Static data types (standard data types, e.g. Boolean, Integer 16, etc.)
0020 – 003F	Complex data types (predefined data structures composed of standard data types)
0040 – 005F	Manufacturer-specific complex data types
0060 – 007F	Device-profile-specific static data types
0080 – 009F	Device-profile-specific complex data types
00A0 – 0FFF	Reserved
1000 – 1FFF	Communication profile area (such as device type, error register, etc.)
2000 – 5FFF	Manufacturer-specific profile area
6000 – 9FFF	Standardized device profile area
A000 – FFFF	Reserved

(c) DeviceNet

DeviceNet uses the CAN standard as the foundation for a higher-level communication protocol, so is often referred to as a CAN application layer protocol. The DeviceNet specification determines the way data are organized and transferred between devices as an object model that devices need to implement. This ensures that all devices present a consistent interface to the rest of the network, hiding their internal details. DeviceNet networks will be the topic of section 10.4 of this chapter; only the linkage between the CAN application layer and the DeviceNet application layer is briefly mentioned here.

(2) Data-link layer

The data-link layer is responsible for transferring messages (or frame) from a given node to all other nodes in the CAN network. This layer handles bit stuffing and checksums for error handling, and after sending a message, waits for acknowledgment from the receivers. It is subdivided into two further layers:

(a) the logical link control layer (LLC), which accepts messages by a filtering process; overload notification and recovery management tasks will be taken care of by this layer;
(b) the medium access control layer (MAC), which carries out data encapsulation, frame coding and arbitration, media access management, error detection and signaling and acknowledgment tasks.

(3) Physical layer

The data-link and physical layers of Figure 10.2, which are normally transparent to a system operator, are included in any controller that implements the CAN protocol with integrated CAN controller. This layer implements: physical signalling, which includes bit encoding and decoding, bit transmit timing and synchronization; identifying the driver and receiver characteristics attached to the CAN bus; and interfacing with connectors.

CAN has several different physical layers which classify certain aspects of the network. The most widely used are described below:

(a) High-speed CAN

High-speed CAN is the most common physical layer. Networks are implemented with two wires and allow communication at transfer rates up to 1 Mb/s. It is also known as CAN C and ISO 11898-2. Typical high-speed CAN devices include anti-lock brake systems, engine control modules, and emissions systems.

(b) Low-speed/fault-tolerant CAN hardware

Low-speed/fault-tolerant CAN networks are also implemented with two wires, they can communicate with devices at rates up to 125 kb/s, and offer transceivers with fault-tolerant capabilities. Other names for low-speed/fault-tolerant CAN include CAN-B and ISO 11898-3. Typical low-speed, fault-tolerant devices in an automobile include comfort devices. Wires that have to pass through the door of a vehicle are low-speed, fault-tolerant to cope with the stress that is imposed when opening and closing a door. Also, in situations where an advanced level of security is desired, such as with brake lights, low-speed, fault-tolerant CAN offers a solution.

(c) Single-wire CAN hardware

Single-wire CAN interfaces can communicate at rates up to 33.3 kb/s (88.3 kb/s in high-speed mode). These interfaces are also known as SAE-J2411, CAN A, and GMLAN. Typical single-wire devices within an automobile are those that do not require high performance, such as seat and mirror adjusters.

(d) Software-selectable CAN hardware

Software-selectable CAN interfaces can often be configured to use any of the onboard transceivers (high-speed, low-speed, fault-tolerant, or single-wire CAN). Multiple-transceiver hardware offers the perfect solution for applications that require a combination of different communications standards. Software-selectable CAN hardware can choose the appropriate external CAN transceiver.

10.1.3 CAN communication

CAN communication is a broadcast mechanism. A transmitting node places data on the network for all nodes to access (Figure 10.4). However, only those nodes requiring updated data allow the data message to pass through a filter. If this filter is not set, much of a node's microprocessing time is spent sorting through messages that are not needed.

(1) CSMA/CD protocol

The CAN communication protocol is a CSMA/CD protocol. CSMA stands for carrier sense multiple access, which means that every node on the network must monitor the bus for a period of no activity before trying to send a message on that bus (carrier sense). Also, once this period of no activity occurs, every node on the bus has an equal opportunity to transmit a message (multiple access). The CD stands for collision detection. If two nodes on the network start transmitting at the same time, the nodes will detect the collision and take the appropriate action. In CAN protocol, a non-destructive bitwise arbitration method is utilized, so messages remain in action after arbitration is completed even if

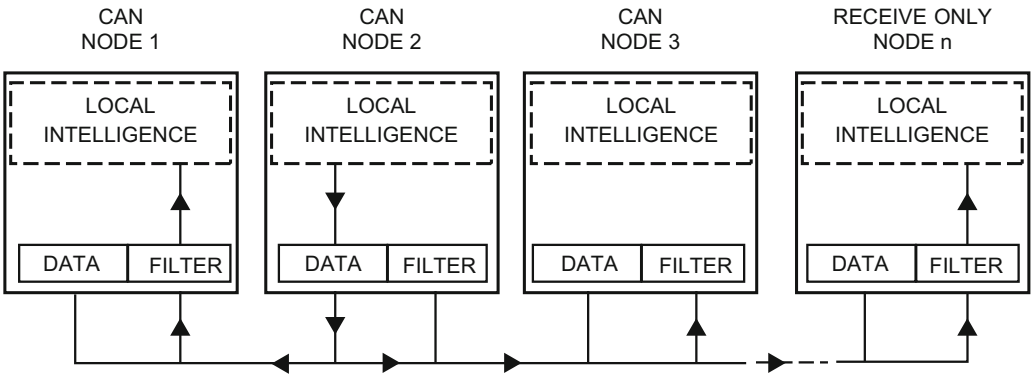


FIGURE 10.4

CAN broadcasting data-flow model.

collisions are detected. All of this arbitration takes place without corruption or delay of the higher-priority message. To support non-destructive bitwise arbitration, logic states need to be defined as dominant or recessive. The transmitting node must also monitor the state of the bus to see whether the logic state it is trying to send actually appears on the bus.

The CAN protocol defines 0 as a dominant bit, and 1 as a recessive bit. A dominant bit state will always be preferred arbitration over a recessive state, therefore the lower the value is in the message identifier (the field used in the message arbitration process; see Tables 10.2 and 10.3), the higher the priority of the message will be. As an example, suppose two nodes are trying to transmit a message at the same time. Each node will monitor the bus to make sure the bit that it is trying to send actually

Table 10.2 Base CAN Frame Format

Field Name	Bits	Function
Start-of-frame	1	Denotes the start of frame transmission
Identifier	11	A (unique) identifier for the data
Remote transmission request (RTR)	1	Must be dominant (0) Optional
Identifier extension bit (IDE)	1	Must be dominant (0) Optional
Reserved bit (r0)	1	Reserved bit (it must be set to dominant (0), but accepted as either dominant or recessive)
Data length code (DLC)	4	Number of bytes of data (0-8 bytes)
Data field	0 - 32	Data to be transmitted (length dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-Frame (EOF)	7	Must be recessive (1)

Table 10.3 Extended CAN Frame Format

Field Name	Bits	Function
Start-of-frame	1	Denotes the start of frame transmission
Identifier A	11	First part of the (unique) identifier for the data
Substitute remote request (SRR)	1	Must be recessive (1) Optional
Identifier extension bit (IDE)	1	Must be recessive (1) Optional
Identifier B	18	Second part of the (unique) identifier for the data
Remote transmission request (RTR)	1	Must be dominant (0)
Reserved bits (r0,r1)	2	Reserved bits (it must be set dominant (0), but accepted as either dominant or recessive)
Data length code (DLC)	4	Number of bytes of data (0 – 8 bytes)
Data field	0 - 32	Data to be transmitted (length dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

appears on the bus. The lower-priority message will at some point try to send a recessive bit and the monitored state on the bus will be dominant. At that point this node loses arbitration and immediately stops transmitting. The higher-priority message will continue to completion, and the losing node will wait for the next period of no activity on the bus to attempt transmission again.

(2) Message-based communication

The CAN protocol is a message-based protocol, rather than an address-based protocol. This means that messages are not transmitted from one node to another node on an address basis. CAN messages comprise the priority and the contents of the data being transmitted. All nodes in the system receive every message transmitted on the bus, and will acknowledge if the message was properly received (Figure 10.4). Each node needs to decide whether the message received should be immediately discarded or kept to be processed. A single message can be destined for one node, or many nodes. A node can also request information from other nodes. This is called a remote transmit request (RTR) node that specifically requests data to be sent to it rather than passively waiting to receive information.

Two examples are given in this paragraph to show how CAN performs message-based communication. Firstly, an automotive airbag sensor is connected via CAN to a safety system router node only. This router node takes in other safety system information and routes it to all other nodes on the safety system network. All the other nodes on the safety system network can then receive the latest airbag sensor information from the router at the same time, acknowledge if the message was received properly, and decide whether to utilize this information or discard it. The second example is a safety system in a car that gets frequent updates from critical sensors such as the airbags, but does not receive frequent updates from other sensors (such as the oil pressure sensor or the low battery sensor) to make

sure they are functioning properly. It can request data periodically from these other sensors and perform a thorough safety system check. The system designer can utilize this feature to minimize network traffic while still maintaining the integrity of the network.

One additional benefit of this message-based protocol is that additional nodes can be added to the system without the necessity to reprogram all other nodes to recognize the addition. The new node will start receiving messages from the network and, based on the message ID, decide whether to process or discard the received information.

The CAN protocol defines four different types of messages (or frames). The first and most common is a data frame, used when a node transmits information to any or all other nodes in the system. Second is a remote frame, a data frame with the RTR bit set so as to signify that it is a remote transmit request. The other two frame types error frames and overload frames are used for error handling. Error frames are generated by nodes that detect any of the protocol errors defined by CAN. Overload errors are generated by nodes that require more time to process messages already received. Table 10.2 is the Base CAN frame format. Table 10.3 is the Extended CAN frame format.

(3) Error handling

CAN nodes can determine fault conditions and change to different modes based on the severity of the problems encountered. They also have the ability to distinguish short disturbances from permanent failures and modify their functionality accordingly. CAN nodes can change from functioning like a normal node (being able to transmit and receive messages normally), to shutting down completely (bus-off) based on the severity of the errors detected. This feature is called fault confinement. No faulty CAN node or nodes will be able to monopolize all of the bandwidth on the network because faults will be confined to the faulty nodes, which will shut off, so preventing network failure. This is very powerful because fault confinement guarantees bandwidth for critical system information.

There are five error conditions that are defined in the CAN protocol and three error states that a node can exist in, based upon the type and number of error conditions detected. The following describes each in more detail.

(a) Error conditions

1. CRC error. A 15-bit cyclic redundancy check (CRC) value is calculated by the transmitting node and transmitted in the CRC field. All nodes on the network receive this message, calculate a CRC and verify that the CRC values match. If the values do not match, a CRC error occurs and an error frame is generated. Since at least one node did not properly receive the message, it is then resent after a proper intermission time.

2. Acknowledge error. In the acknowledge field of a message, the transmitting node checks whether the acknowledge slot (which it has sent as a recessive bit) contains a dominant bit. This dominant bit would acknowledge that at least one node correctly received the message. If this bit is recessive, then no node received the message properly, and an acknowledge error has occurred. An error frame is then generated and the original message will be repeated after a proper intermission time.

3. Form error. If any node detects a dominant bit in one of the following four segments of the message: end of frame, interframe space, acknowledge delimiter or CRC delimiter, the CAN protocol defines this as a form violation and a form error is generated. The original message is then resent after a proper intermission time. (Please see Table 10.2 and/or Table 10.3 for where these segments lie in a CAN message.)

4. Bit error. A bit error occurs if a transmitter sends a dominant bit and detects a recessive bit, or if it sends a recessive bit and detects a dominant bit when monitoring the actual bus level and comparing it to the bit that it has just sent. In the case where the transmitter sends a recessive bit and a dominant bit is detected during the arbitration field or acknowledge slot, no bit error is generated because normal arbitration or acknowledgment is occurring. If a bit error is detected, an error frame is generated and the original message is resent after a proper intermission time.

5. Stuff error. CAN protocol uses a non-return-to-zero (NRZ) transmission method. This means that the bit level is placed on the bus for the entire bit time. CAN is also asynchronous, and bit stuffing is used to allow receiving nodes to synchronize by recovering clock information from the data stream. Receiving nodes synchronize on recessive to dominant transitions. If there are more than five bits of the same polarity in a row, CAN issues automatically stuff an opposite polarity bit in the data stream. The receiving node(s) will use it for synchronization, but will ignore the stuff bit for data purposes. If, between the start of frame and the CRC delimiter, six consecutive bits with the same polarity are detected, then the bit stuffing rule has been violated. A stuff error then occurs, an error frame is sent, and the message is repeated.

(b) Error states

Detected errors are made public to all other nodes via error frames or error flags. The transmission of an erroneous message is aborted and the frame is repeated as soon as the message can again win arbitration on the network. Each node will be in one of three error states at this point; error-active, error-passive or bus-off.

1. Error-active. An error-active node can actively take part in bus communication, including sending an active error flag (which consists of six consecutive dominant bits). The error flag actively violates the bit stuffing rule and causes all other nodes to send an error flag, called the error echo flag, in response. An active error flag and the subsequent error echo flag may cause as many as twelve consecutive dominant bits on the bus; six from the active error flag, and zero to six more from the error echo flag depending upon when each node detects an error on the bus. A node is error-active when both the transmit error counter (TEC) and the receive error counter (REC) are below 128. Error-active is the normal operational mode, allowing the node to transmit and receive without restrictions.

2. Error-passive. A node becomes error-passive when either the transmit error counter or receive error counter exceeds 127. Error-passive nodes are not permitted to transmit active error flags on the bus, but instead, transmit passive error flags which consist of six recessive bits. If the error-passive node is currently the only transmitter on the bus then the passive error flag will violate the bit stuffing rule and the receiving node(s) will respond with error flags of their own (either active or passive depending upon their own error state). If the error-passive node in question is not the only transmitter (i.e. during arbitration) or is a receiver, then the passive error flag will have no effect on the bus due to the recessive nature of the error flag. When an error-passive node transmits a passive error flag and detects a dominant bit, it must see the bus as being idle for eight additional bit times after an intermission before recognizing the bus as available. After this time, it will attempt to retransmit.

3. Bus-off. A node goes into the bus-off state when the transmit error counter is greater than 255 (receive errors cannot cause a node to go bus-off). In this mode, the node cannot send or receive messages, acknowledge messages, or transmit error frames of any kind. This is how fault confinement is achieved. There is a bus recovery sequence defined by the CAN protocol that allows a node that is bus-off to recover, return to error-active, and begin transmitting again if the fault condition is removed.

10.1.4 CAN bus

The CAN bus is a multiplexed wiring system used to connect the ECU as nodes to the CAN network.

(1) CAN bus I/O interface

The CAN standard defines a communication network that links all the nodes connected to a bus and enables them to communicate with one another. There may or may not be a central control node, and nodes may be added at any time, even while the network is operating (hot-plugging). All the nodes are connected to the network by interface devices that can be CAN transceivers or CAN connectors. These interface devices commonly have the following parameters specified: supply voltage; high short-circuit protection; bus-input impedance; wide common-mode range; thermal shutdown protection; low current standby and sleep mode; controlled transition time; etc.

(2) Bus length and signaling rate

The basics of arbitration require that the front wave of the first bit of a CAN message should travel to the most remote node on a CAN network and back again before the bit is designated by the receiver of the sending node as dominant or recessive (typically this sample is made at about two-thirds the bit width). With this limitation, the maximum bus length and signaling rate are determined by network parameters. Factors to be considered in network design include the (approximately) 5 ns/m propagation delay of a typical twisted-pair bus cable, signal amplitude loss due to the loss mechanisms of the cable, and the input impedance of the bus transceivers. Under strict analysis, variations among the different oscillators in a system also need to be accounted for by adjustments in signaling rate and bus length. Maximum signaling rates achieved with the SN65HVD230 in high-speed mode with several bus lengths are listed in Table 10.4.

(3) Maximum number of nodes

In practice, up to 64 nodes may be connected to a DeviceNet bus, 127 on a CANopen bus and up to 255 nodes may be connected together on a CAN bus. When more than the standard 30 nodes are used on a bus, it is recommended that a transceiver (or connector) with high bus-input impedance be used.

A problem may develop when too many transceivers (or connectors) source or sink current from or onto the bus. When a device begins to transmit a message, it has to sink or source all of the leakage current on the bus, plus drive the standard signal voltage levels across the termination resistance. If the current demand is too great, the device may be driven into thermal shut-down or even destroyed.

Table 10.4 Suggested CAN Cable Length and Signaling Rate

Bus Length (m)	Signalling Rate (Mbps)
40	1
100	0.5
200	0.25
500	0.10
1000	0.05

To prevent transceiver (or connector) damage, the voltage difference between reference grounds of the nodes on a bus should be held to a minimum. This is the common-mode voltage across the entire system and although many transceivers (or connectors) are designed to operate over an extended common-mode range, the cumulative current demand of too many devices at a common-mode voltage extreme may jeopardize network security. To enhance this common-mode security, higher-layer protocols such as DeviceNet specify that power and ground wires be carried along with the signaling pair of wires. Several cable companies have developed four-wire bundled cables specifically for these applications.

10.2 SUPERVISORY CONTROL AND DATA ACQUISITION (SCADA) NETWORK

10.2.1 Introduction

Supervisory control and data acquisition (SCADA) networks are large-scale industrial control and measurement systems primarily used to control and monitor the condition of field-based assets remotely from a central location. SCADA networks are used pervasively in a wide range of industries, including electricity power generation and transmission, chemical and petrochemical industries, highways and transport, oil and gas and steel production, as well as research and development. Modern SCADA systems exhibit predominantly open-loop control characteristics and utilize predominantly long-distance communications, although some elements of closed-loop control and or short-distance communications may also be present. A SCADA system performs four functions.

(1) Data acquisition

The systems to be monitored by SCADA generally have hundreds or thousands of field devices (or field instrumentation) such as sensors, actuators, switches, valves, transmitters, meters and drivers. Some devices measure inputs into the system (for example, water flowing into a reservoir), and some measure outputs (such as valve pressure as water is released from the reservoir). They may measure simple events that can be detected by a straightforward on-off switch, called a digital input. In industrial control applications, such digital inputs are used to measure simple states. Alternatively, they may measure more complex situations where exact measurement is important. Analog devices, which can detect continuous changes in a voltage or current input are used for these cases. They can track fluid levels in tanks, voltage levels in batteries, temperature and any other factor that can be measured in a continuous range of input. For most analog factors, there is a normal range defined by a bottom and top level.

Modern SCADA systems usually require real-time data acquisition i.e. data must be collected in such a way that it can be processed or stored by a computer instantly. Data acquisition normally involves an input scanner or switch, an analog-to-digital converter, and signal conditioners that can be either energy sensors or conditioning processes so that the data signals directly in engineering units are transmitted. Data acquisition systems can also form part of a process control system which, through the use of appropriate software, provides direct digital control of an industrial process. They can also be used for data logging and process or alarm monitoring.

(2) Data communication

Many control applications need to monitor multiple systems from a central control station, thus necessitating a communications network to transport all the data collected from the field

instrumentation. Early SCADA networks communicated over radio, modem or dedicated serial lines, but currently most systems put SCADA data on Ethernet or over the Internet. Real SCADA data are encoded into protocol format. Older SCADA systems depended on closed proprietary protocols, but today the trend is towards open, standard protocols and protocol mediation. The remote telemetry (or terminal) unit (RTU) or the programmable logic controller (PLC) is needed to provide an interface between the field instrumentation and the SCADA network. The RTU or PLC encodes field inputs into protocol format and forwards them to the SCADA master station; in turn, the RTU receives control commands in protocol format from the master, and transmits electrical signals to the appropriate control relays and then downward to field instrumentations.

(3) Data presentation

The SCADA system continuously monitors all field instrumentation and alerts the operator when there is an "alarm", that is, when a control factor is operating outside what is defined as its normal range. The master presents a comprehensive view of the entire managed system, and can present more detail in response to user requests. A human machine interface (HMI) presents process data to a human operator, and allows the human operator to control the process. An HMI may also be linked to a database, to provide trending, diagnostic data, and management information such as scheduled maintenance procedures, logistic information, or detailed schematics. SCADA is popular due to its compatibility and reliability. Its use ranges from small applications, such as the control of the temperature of a room, to extremely large applications, such as the control of nuclear power plants.

(4) System control

SCADA systems can regulate all kinds of industrial processes automatically. For example, if too much pressure is building up in a gas pipeline, a SCADA system could automatically open a release valve. Electricity production can be adjusted to meet demands on the power grid. Even these real-world examples are simplified; a full-scale SCADA system can adjust a managed system in response to multiple inputs.

SCADA systems generally cover large geographic areas with the controller application housed in the appropriate terminal that is controlled by an operator working centrally. Reliable communication links between the SCADA central host and the field-level devices are therefore crucial to the efficient operation of such systems. Where a critical control algorithm is required and the controller must be located remotely, the communication link must be designed to contribute effectively to the reliability of the entire system. The cost associated with this requirement may be high enough to warrant placing the automatic control function at the site.

SCADA systems are coming in line with standard networking technologies, with Ethernet and TCP/IP-based protocols replacing the older proprietary standards. Although certain characteristics of frame-based network communication technology (determinism, synchronization, protocol selection, environment suitability) have restricted the adoption of Ethernet in a few specialized applications, it is now broadly accepted in the majority of situations. With the emergence of software as a service in the broader software industry, some vendors have begun offering application-specific SCADA systems hosted on remote platforms over the Internet. This removes the need to install and commission systems at the end-user's facility and takes advantage of security features already available in Internet technology. Some concerns inherent in this approach include security, Internet connection reliability, and latency.

SCADA systems are becoming increasingly ubiquitous. Thin clients, web portals, and web-based products are gaining popularity with most major vendors.

10.2.2 SCADA components and hardware

SCADA encompasses the transfer of data between a central host or master station and a number of remote field sites. System architecture can range from simple local control to a large-scale distributed control system. Figure 10.5 shows a generic SACDA system that consists of a central host or master terminal unit (MTU) with HMI; a number of the RTUs which connect with the field-level instrumentations; and networks for data communication between the MTU and RTUs.

The overall form of a SCADA system, is for a central host or MTU to receive a field data from the RTUs. The received data are then processed by central software. Once this processing is complete, the central host then logs alarms and displays the data on its HMI screens.

(1) Central host or master station or MTU

The central host or master station or MTU is a single computer or a cluster of computer servers that provide an interface to the SCADA system for human operators that processes the information received from the RTUs and presents the results in a human-readable form. The host computer then sends commands to the RTU sites for use by field instrumentation.

The primary operator interface is a graphical user interface (GUI) showing a representation of the plant or equipment. Live data are shown as graphical shapes (foreground) over a static background. As the data change in the field, these shapes are updated. For example in a water supply system, a valve may be shown as open or closed, depending on the latest digital value from the field instrumentations.

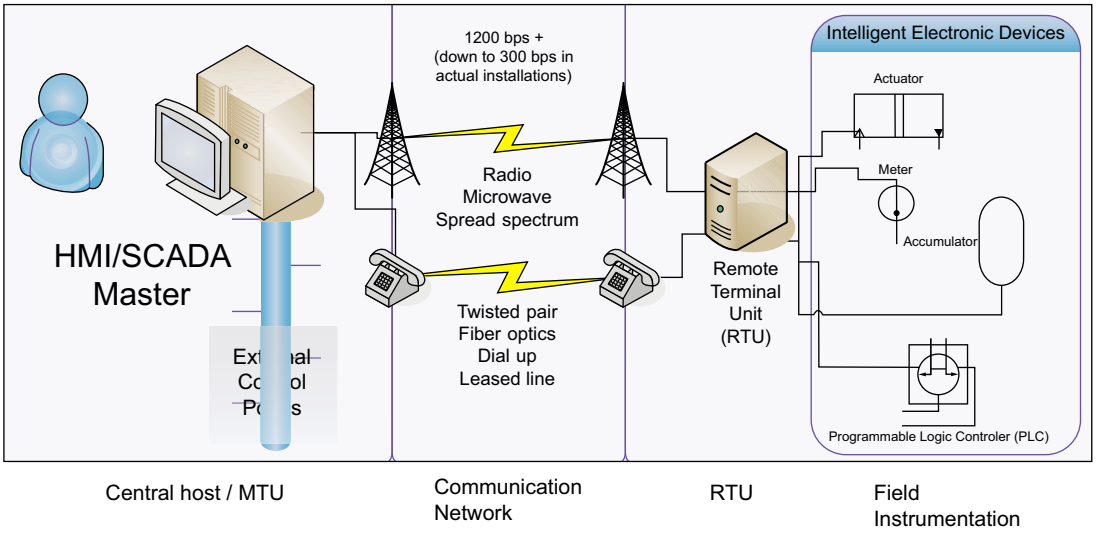


FIGURE 10.5

A modern SCADA system architecture.

The most recent analog values are displayed on the screens as numerical values or as some physical representation. Alarms may be represented on a screen as a red flashing icon above the relevant field device. The system may have many such displays, and the operator can select from the relevant ones at any time.

If an alarm is triggered, it will be displayed on dedicated alarm lists. Any abnormal conditions that are detected in the field instrumentation are registered at the central host as alarms, and operators are notified, usually by an audible alert and visual signals. Historical records of each alarm and the name of the operator who acknowledged the alarm can be held within a self-contained archive for later investigation or audit requirements. Where variables change over time, the SCADA system usually offers a trending system to follow its behavior.

(2) Field data interface devices

These devices provide information about how the field instruments are running. To be useful, the information that is passed to and from the field data interface devices must be converted to a form that is compatible with the language (or protocol) of the SCADA system. To achieve this, some form of electronic field data interface is required.

(a) Remote terminal units (RTUs)

RTUs are primarily stand-alone data acquisition and control units. A typical RTU configuration is shown in Figure 10.6, which includes the hardware modules such as control microprocessor and associated memory, analog inputs, analog outputs, counter inputs, digital inputs, digital outputs, communication and I/O interfaces, power supply, together with an RTU rack and enclosure. Small RTUs generally have fewer than 10 to 20 analog and digital signal inputs; medium-sized RTUs

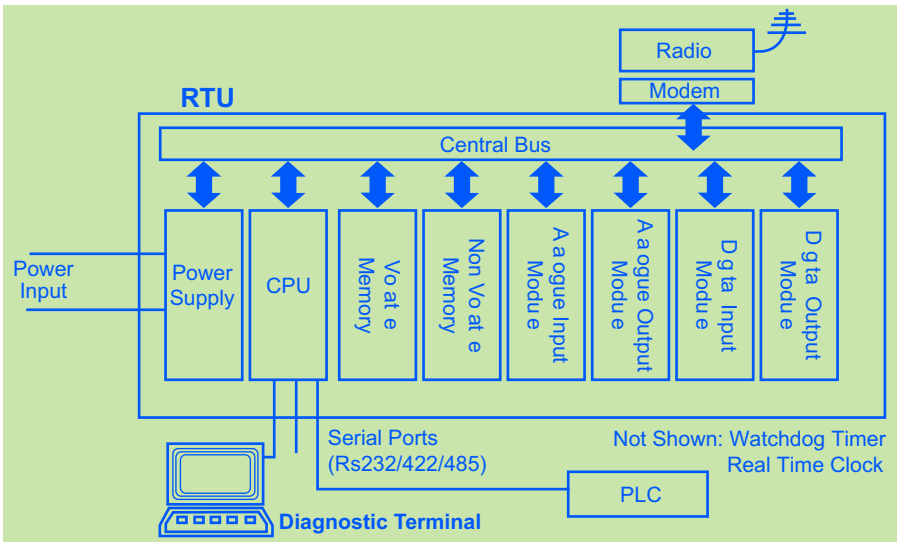


FIGURE 10.6

Typical RTU hardware configuration in SCADA systems.

have 100 digital and 30 to 40 analog signal inputs. Any RTU with more signal inputs than this is referred to as large.

RTUs are used to convert electronic signals received from (or required by) field instrumentation into (or from) the communication protocol that is used to transmit the data over a network. RTUs appear in the field as a box in a switchboard with electrical signal wires running to field instrumentation and a cable linked to a communication channel interface, such as a telephone cable, a field bus, or a radio.

The instructions for the automation of field data interface devices are usually stored locally since the limited bandwidth typical of communication links to the SCADA central host makes central storage impractical. Such instructions are traditionally held within local electronic devices known as programmable logic controllers (PLCs), which have in the past been physically separate from RTUs. PLCs connect directly to field instruments and incorporate programmed intelligence in the form of logical procedures that will be executed in the event of certain field conditions. More recent systems have no PLCs and local control logic is held within the RTUs, or in relay logic in the local switchboard.

(b) Programmable logic controllers (PLCs)

PLCs have their origins in industrial automation and therefore are often used in manufacturing and process plant applications. SCADA systems, on the other hand, have origins in early telemetry applications, where it was only necessary to receive basic information from a remote source. The RTUs connected to these systems had no need for control programming because the local control algorithm was held in the relay switching logic.

As PLCs and telemetry have become more commonly used, the need to influence the program within the PLCs via a remote signal has arisen. This is in effect the "supervisory control" part of the SCADA acronym. A simple local control program could be stored within a RTU and perform the control within that device. At the same time, PLCs started to include communications modules to allow reporting of the state of the control program. PLC and RTU manufacturers therefore compete for the same market.

As a result of these developments, the distinction between PLCs and RTUs has blurred and the terminology is virtually interchangeable. For the sake of simplicity, the term RTU will be used to refer to a remote field data interface device; however, such a device could include automation programming that traditionally would have been classified as a PLC.

(3) Communication networks

Most industrial applications use networks to enable communication between the central host and the RTUs. Control actions performed at the central host are generally treated as data that are sent to the RTUs, so any control action will initiate a communication link with the RTUs to allow the command to be sent to the field device. Modern SCADA systems usually employ several layers of checking mechanisms to ensure that a transmitted command is received by its intended targets.

There are two commonly used communication models in the SCADA networks: a polled approach and a contention approach.

(a) Polled communication model (master-slave)

This can be used in a point-to-point or multi-point network configuration and is probably the simplest communication model to use. The master is in total control of the communication system and makes

repetitive requests for data to be transferred to and from each one of a number of slaves. The slaves do not initiate the transactions but rely on the master. The approach is essentially half-duplex. If a slave does not respond in a defined time, the master then retries (typically up to three times) and then marks the slave as unserviceable before trying the next slave node in the sequence. It is possible to retry the unserviceable slave again on the next cycle of polling. Figure 10.7 illustrates polling communication models between the SCADA master and SCADA RTUs/PLCs. Figure 10.7 is an automatic meter reading SCADA system, which uses the POTS telephone service of TC2800 RS-232 and RS-422 and RS-485 multidrop fiber-optic multiplexer, and the TC1900 RS-232 telephone extender.

(b) Contention communication model (peer-to-peer)

A contention method such as carrier sense with multiple access/collision detection (CSMA/CD) can be used for communications control. There is no controlling master, and individual stations have to contend (complete) for access to the transmission medium. In such an arrangement, collisions are unavoidable and stations have to deal with them. In a situation where a RTU wants to communicate with another one, a technique used is to respond to a poll by the master station with a message with a destination address other than that of the master station. This approach can be used either in a master-slave network or a group of stations all having equal status. Collisions are avoided by listening to the medium before transmitting. The systems rely on recovery methods to handle collision problems.

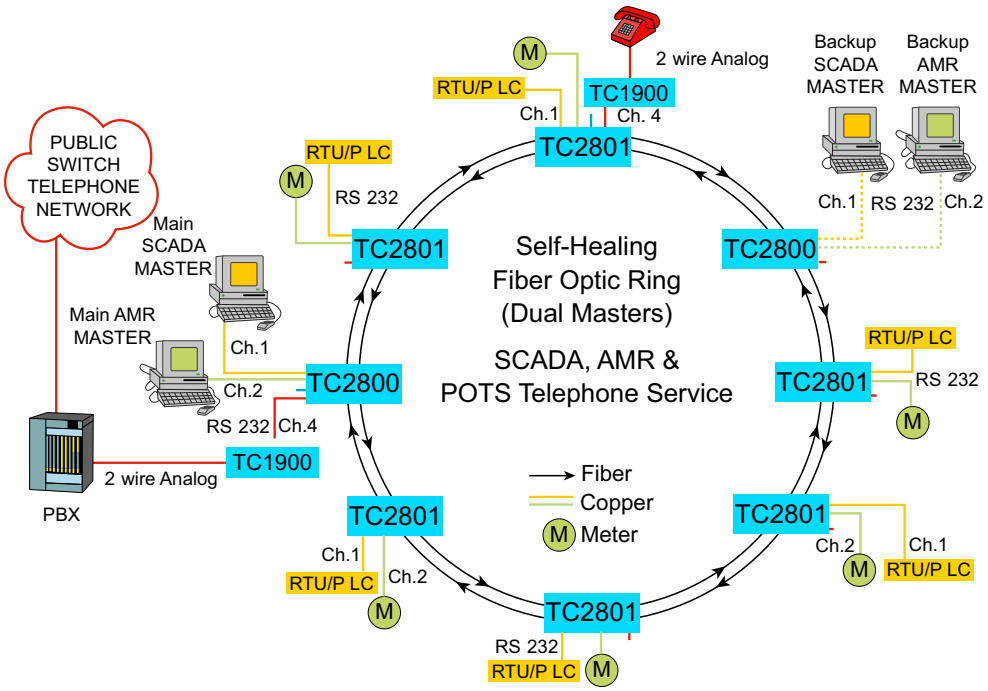


FIGURE 10.7

A polling semantics applied for the communication between master station and RTUs/PLCs in SCADA networks.

Typically these systems are very effective at low capacity rates; but as soon as the traffic rises to over 30% of the channel capacity there is an avalanche-type collapse of the system and communication becomes unreliable and erratic. The technique is used solely on networks where all RTUs/PLCs have access to the same media within radio range or on a common cable link.

Unnecessary transfer of data is reduced by exception reporting. This approach is popular with the CSMA/CD method but it could also offer a solution for the polled approach, where there is a considerable amount of data to transfer from each slave. The remote station monitors its own inputs for a change of state or data. When this occurs, the remote station writes a block of data to the master station. Each analog or digital point to be reported back to the central master station has a set of exception reporting parameters associated with it, such as high and low alarm limits for individual analog values.

A practical approach to combining the master slave with CSMA/CD is to use the concept of a slot time for each station. Assume that the architecture includes a master and a number of slaves that need to communicate with the master station. No communication between slaves is required (except for possibly through the master). The peroid of time over which each station is allowed to transmit is called a slot time. There are two types of slots; a slave or a few slaves transmitting to a master, and a master transmitting to a slave. A slot time is calculated as the sum of the maximum of modem setup time (for example, 30 milliseconds) plus radio transmit time (for example, 100 milliseconds) plus time for protocol message to transmit (for example, 58 milliseconds) plus muting time (for example, 25 milliseconds) for each transmitter. The master commences operations by polling each slave in turn. Each slave will synchronize on the master message and will transmit an acknowledge message. Hereafter, slaves will only transmit by using CSMA/CD during the master receiving time slots, which alternate with the master transmission time slots. Once a slave node detects a state change, it will transmit the data on the first available master receive time slot. If two remote slaves try to transmit in the same time slot, the message will be corrupted and the slaves will not receive a response from the master. The slaves will then randomly select a subsequent master receiver time slot and attempt a retransmission of the message. If the master continues to get corrupted messages, it may elect to do a complete poll of all remote slaves as the CSMA/CD type mechanism is possibly breaking down due to excessive traffic.

(4) Field instrumentation

These refers to those devices that are connected to equipment being controlled and monitored, which convert physical parameters to electrical signals at the field level. Typical field instrumentations are: sensors, actuators, switches, transmitters, meters, detectors, relays, valves and drivers.

Data acquisition within SCADA systems is accomplished by the RTUs first scanning the connected field instrumentation. The time to perform this task is called the scanning interval and can be less than two seconds. The central host computer scans the RTUs (usually at a much slower rate) to access the data in a process referred to as polling the RTUs. Some systems allow the RTU to transmit field values and alarms to the central host without being polled by the central host. This mechanism is known as unsolicited messaging. Systems that allow this mechanism usually use it in combination with the process of polling the RTU to solicit information as to the health of the RTU. Unsolicited messages are usually only transmitted when the field data have deviated by a pre-specified percentage, so as to minimize the use of the communications channels, or when a suitably urgent alarm indicating some site abnormality exists.

10.2.3 SCADA software and firmware

An important component of every SCADA system is the computer software used within the system. Depending on the size and nature of the application, SCADA software can be a significant cost item. When software is well-defined, designed, written, checked, and tested, a successful SCADA system is likely to result.

Much SCADA is software configured for a specific hardware platform. A wide range of general SCADA software products are also available, some of which may suit the required application. These are usually more flexible, and compatible with different types of hardware and software. It is therefore important to select the software systems appropriate to any new SCADA system.

SCADA systems typically implement a distributed database which contains data elements called tags or points. A point represents a single input or output value monitored or controlled by the system. Points can be either "hard" or "soft". A hard point represents an actual input or output within the system, while a soft point results from logic and mathematical operations applied to other points. Points are normally stored as value-timestamp pairs, i.e. a value and the timestamp when it was recorded or calculated. A series of value-timestamp pairs gives the history of that point. It is also common to store additional metadata with tags, such as the path to a field device or register in an interface device to field level, design time comments, and alarm information.

SCADA software products provide the building blocks for the application-specific software, which must be defined, designed, written, tested, and deployed separately for each system. Such software products are used within the following components of a SCADA system:

(a) The central host computer operating system, used to control the central host. This can be based on UNIX, Windows NT or other operating system.

(b) The operator terminal operating system, which is also used to control the central host computer hardware, is usually the same type as the central host computer operating system. It usually deals with the networking of the central host to the operator terminals.

(c) The central host computer application, which is the software responsible for handling the transmission and receiving of data between the RTUs and the central host. This software usually includes some expert systems defined for application-specific functions. It also provides a graphical user interface offering site mimic screens, alarm pages, trend pages, and control functions.

(d) The operator terminal application software, which enables users to access the information on the central host computer application. This software can also include some expert systems defined for application-specific functions. It is usually a subset of the software used on the central host computers.

(e) Communication protocol drivers, which are usually based within the central host and the RTUs, and are required to control the translation and interpretation of the data between ends of the communications links in the system. The protocol drivers prepare the data for use either at the field devices or at the central host end of the system. The communication models and protocols for the SCADA system will be discussed in subsection 10.2.4.

(f) Communications network management software, which is required to control the communications network and to allow it to be monitored for attacks and failures. SCADA communication security and system reliability will be the topic of subsection 10.2.5.

(g) RTU automation software, which allows engineering staff to configure and maintain the application housed within the RTUs (or PLCs). This often includes the local automation application and any data processing tasks that are performed within the RTUs (or PLCs).

10.2.4 SCADA communication protocols

In 1988, the International Electrotechnical Commission (IEC) began publishing a standard entitled "IEC 870 Telecontrol Equipments and Systems", of which one part was "Part 5 Transmission Protocols". This part was developed in a hierarchical manner and published in a number of sub-parts, taking from 1990 to 1995 to completely define an open protocol for SCADA communications. The protocol was defined in terms of the open systems interconnection model (OSI) using a minimum subset of the layers: the physical layer, data-link layer, and application layer. This protocol standard included the definition of message format at the data-link layer, and a set of data formats at the application layer so that it could be used to create systems capable of interoperation.

This IEC standard was subsequently renumbered with a prefix of 60, and so the current IEC standard for transmission protocols is IEC 60870-5. The IEC 60870-5 was defined primarily for telecommunication of control information within electrical power systems, so having data formats specifically related to this application. Although it includes general data types that could be used in any SCADA application, the use of IEC 60870 has since been mostly confined to the electricity power industry.

During the same period, the protocol entitled Distributed Network Protocol Version 3.3 (DNP3) was developed by Harris Control Division of Distributed Automation Products and released to the industry-based DNP3 User Group in November 1993. DNP3 has gained significant acceptance since then, both geographically and over different industries. DNP3 is supported by a large number of vendors and users in electrical power, water infrastructure, and other sectors in America, Africa, Asia, Australia and New Zealand. In Europe, DNP3 competes with IEC 60870-5, but the latter is confined to the electrical power grid industry, whereas DNP3 has found wider application in the energy, water, and security industries, and so on.

Both protocols were designed specifically for SCADA applications, and were thus widely supported by manufacturers of SCADA systems. They involve acquisition of information and sending of control commands between physically separate computer devices. They are designed to transmit relatively small packages of data in a reliable manner, with the messages arriving in a deterministic sequence. In this respect, both DNP3 and IEC 60870-5 differ from general purpose protocols such as file transfer protocol (FTP) because FTP can send large files, but in a way that is generally not as suitable for SCADA control.

Besides DNP3 and IEC 60870 protocols, there is little push for utility communications architectures (UCA), although the concepts are likely to become routine in the SCADA industry. In 1999, the IEEE published UCA Version 2, as an IEEE standard to address issues that were identified in field testing of the original specification, and to embrace the Internet suite of protocols, which had become widely accepted since the publication of UCA Version 1. However, it is envisaged that DNP3 and UCA will complement each other in the near future.

(1) DNP3 protocols

DNP3 is a telecommunications standard that defines communications between a master station and other intelligent electronic devices. DNP3 supports multiple-slave, peer-to-peer and multiple-master communications with possible polled and quiescent operational modes. Quiescent operation, referred to as reporting by exception, is so called because polls to check for changes are not required. This is because the master station can rely on the outstation to send an "unsolicited response" when it has

a change that needs to be reported. In a quiescent system, generally a periodic background poll is still used, perhaps at hourly intervals, to guard against undetected communications failure. If this was not done, the master station would have no way of detecting a communications failure with the outstation.

The capability to support peer-to-peer and quiescent operation requires that stations which are not designed as master stations are able to initiate communications. This is sometimes referred to as balanced communications, meaning that any station can act as a primary (or sending) and as a secondary (responding) station at the same time. Despite the ability for slave (non-master) stations to initiate communications within DNP3, only master stations can initiate requests for data, or issue commands, to other stations. Thus, although the term balanced is applied to the communications system, the differentiation between master and slave stations remains. Sometimes the terms master and outstation are more applicable.

Architectures may also use protocol converters, for instance in the case of a hierarchical topology, where the outstation devices only use DNP3, and the SCADA master station might use a different communications system. In the case of DNP3 devices with a network port, DNP3 is encapsulated within TCP/IP Ethernet packages. Although this does add an overhead, it does provide an effective means of using existing corporate network to accommodate the SCADA system.

DNP3 provides time-stamping of events with resolution of events down to one millisecond. For events to match up correctly across a system, it is essential that the clocks in all outstations are synchronized with that of the master station. Synchronizing of an outstanding clock is done by sending a time and date object (object 50, variation 1) to the outstation. There is, however, a finite delay in transmission from the master to the outstation, and if this is not accounted for when setting the clock, its time would be in error by this transmission time. This error can be increased by store-and-forward delays introduced by a variety of devices along the transmission path, including (a) time in modern buffers; (b) time in data radio buffers; (c) time in intermediate repeater store-and-forward buffers.

In addition to these, it is also necessary to add the possibility of processing delays between the application level and the lower levels, or "stack delays" at both master and outstation. The data-link functionality is required to record the time of transmission or receipt of the first bit of any delay time message. When transmitting the message, the data-link layer triggers a freeze of the master station clock, which is stored as "master send time". Similarly, when a slave station receives a delay time message it must store a freeze of the local clock as "RTU receive time".

DNP3 provides for measurement of the round trip delay time by the procedure defined as "Function Code 23 Delay Measurement". This procedure follows these steps:

(a) master sends "Function Code 23 Delay Measurement";
(b) master records the start of transmission time as "Master send time";
(c) outstation records the receipt time as "RTU receive time";
(d) outstation commences to send a response and records this time as "RTU send time";
(e) outstation calculates its internal processing turnaround time "RTU turn around" which is equal to the difference by minus the "RTU receive time" from the "RTU send time";
(f) outstation includes in the response a time delay object (object 52, variation 1 or 2) having the value of the turnaround time that is "RTU turn around";
(g) master freezes its clock as "Master receive time" once receiving the response.

The master station now calculates the one-way propagation delay by this formula: Delay = ((“master send time” “Master receive time” “RTU turn around) / 2). This delay time is now adjusted by the following steps:

(a) master sends a write-request containing the time and date objects (object group 50, variation 1); the time value is the master clock time at commencing to send this message “Master send” plus the calculated delay;
(b) outstation receives the first bit of the message at time “RTU receive”;
(c) outstation now sets the outstation time by the following calculation: (“Adjustment” = “Current RTU time” “RTU receive”) and (“New RTU time” = “Time in time and date object” + “Adjustment”).

To obtain interoperability between different manufacturers’ versions of a protocol, it is necessary to ensure that all implementations support the same data objects and functions. This would require standardization of a significant subset of data object types, and an RTU interfacing to a number of devices could require yet more. In DNP3, these issues are addressed through the definition of three subset levels.

(a) DNP3-L1 is layer 1, which gives the simplest level of DNP implementation. It is intended for use between a master station or intermediate device such as a data concentrator, and a small intelligent electronic device such as a relay, a meter, or a stand-alone controller. Normally any inputs and outputs are local to the device.
(b) DNP3-L2 is layer 2, which defines a larger subset of DNP features than the level 1. It is intended to lie between a master station or data concentrator, and an RTU or a large intelligent electronic device. Normally any inputs and outputs are local to the device.
(c) DNP3-L3 is layer 3, which defines the largest subset of DNP features. It does not require support of all the possible DNP features, but it covers those most frequently required. This is implemented typically between a master station and a large advanced RTU. The inputs and outputs of subset level 3 devices would typically be remote as well as local.

(2) IEC 60870-5 protocols

IEC 60870-5 is a collection of standards produced by the International Electrotechnical Commission (IEC) to supply an open standard for the transmission of SCADA telemetry control and information. This standard provides a detailed functional description for telecontrol equipment and systems for controlling geographically widespread processes, which are mostly SCADA systems. Although it was primarily aimed at systems in electrical power industries, is also intended for general SCADA applications. IEC 60870-5 has been merged with TCP/IP to provide a communication protocol over standard computer networks.

The IEC 60870 standard is structured in a hierarchical manner, comprising six parts plus a number of companion standards. Each part is made up of a number of sections, each of which has been published separately. In addition to the main parts, there are four companion standards providing the details of the standard for a particular field of application. These companion standards extend the main definition by adding information objects specific to a given field of application.

The structure of the IEC 60870 standard is illustrated in Figure 10.8. This shows the main parts, the sections, and companion standards connected to transmission protocols. The companion standards are

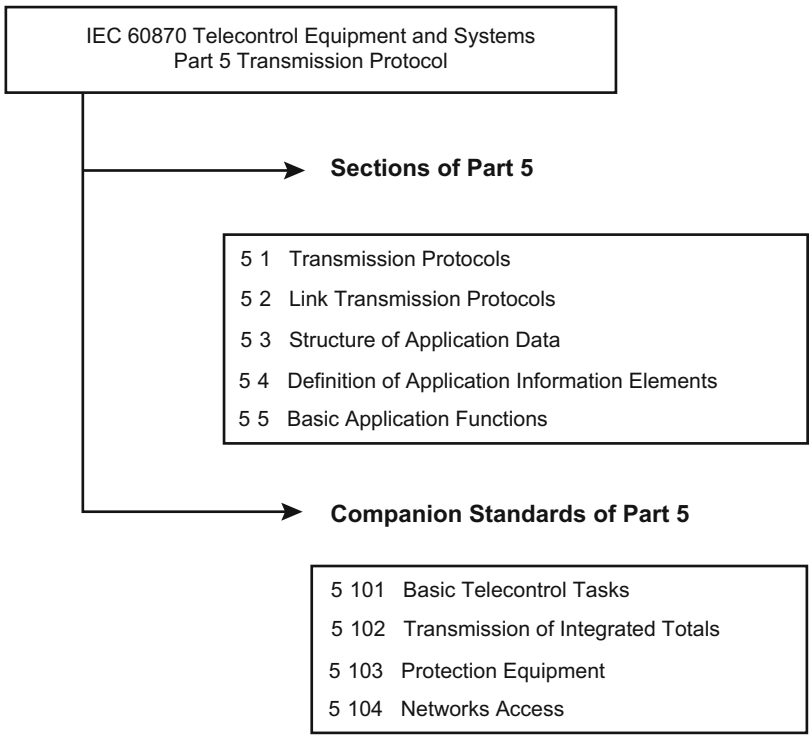


FIGURE 10.8

Structure of IEC 60870 Standard.

publishing in IEC-60870-5-101 to IEC 60870-5-104. IEC 60870-5-101 completes the definition for the transmission protocol. IEC 60870-5-4 defines the transport of IEC 60870-5 application messages over networks. Therefore, the key point to note is that this standard has been progressively developed and issued from IEC-60870-5-101 to IEC 60870-5-104.

(a) System topology

IEC 60870-5-101 supports point-to-point and multidrop communication links carrying bit-serial and low-bandwidth data communications. It allows use of either balanced or unbalanced communication. With unbalanced communication, only the master can initiate communications by transmitting primary frames. This simplifies system design because there is no need to support collision avoidance.

IEC 60870-5 assumes a hierarchical structure in which any two stations communicating with each other, have one designated as the controlling station, and the other as the controlled station. This is also a defined "monitor direction" and a "control direction". Thus monitored data such as analog values from the field are sent in the monitoring direction, and commands are sent in the control direction. If a station sends both monitored data and commands, it is acting as both a controlled and a controlling station. This is defined as dual-mode operation, which is accommodated by the protocol, but requires that use is made of originator addresses.

All communications are initiated by master station requests. Although IEC 60870-5-101 supports balanced communication, this is limited to point-to-point links only. Support for unsolicited messages from a slave does not cover multidrop topology and must employ a cyclic polling scheme to interrogate the secondary stations.

In summary, balanced communication can: (1) initiate a transaction; improve the efficiency of communication system usage; (2) encounter collision problems because two stations can transmit their messages simultaneously; thus collision avoidance and recovery are required; (3) perform the communications limited to point-to-point only.

Unbalanced communication can: (1) send primary messages by master station only; (2) perform communication between multidrops; (3) have no collision while two or more stations transmit their messages at the same time; thus collision avoidance is not required; (4) have a simpler data-link layer function on the slave side in the service data unit of the application layer.

(b) Message structure

The IEC 60870-5-101 message structure is formed by a data-link layer frame carrying link address and control information, a flag indicating whether Class 1 data is available, and optional application data. Each frame can carry a maximum of one application service data unit for the application layer. Figure 10.9(A) shows the data-link frame structure, and the structure of the service data unit of the application layer.

In the case where user data is not required in the frame, either a fixed-length frame or a single character acknowledgment may be used. These provide for efficient use of communications bandwidth. Figure 10.9(B) shows a fixed-length frame and a single control character.

(c) Addressing

Under IEC 60870-5-191, addressing occurs both at the link and at the application level. The link address field may be 1 or 2 octets for unbalanced and 0, 1 or 2 octets for balanced communications. As

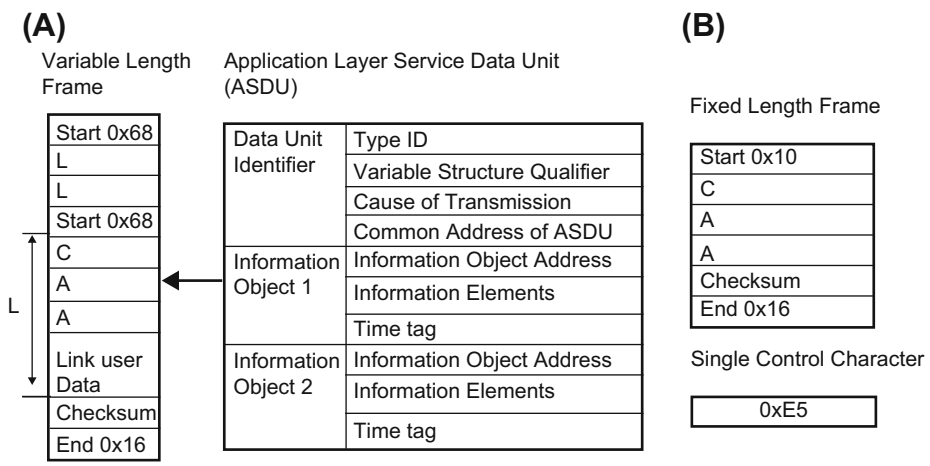


FIGURE 10.9

(A) Message structure for IEC 60870-5-101. (B) Fixed-length frame and single control character.

balanced communications are point-to-point, the link address is redundant, but may be included for security. The link address FF or FFFF is defined as a broadcast address, and may be used to address all stations at the link level. At the application level, the service data unit in this layer contains a 1 or 2 octet common address. This is defined as the address of the controlling station in the control direction, and the address of the controlled station in the monitoring direction. The common address of the service data unit combined with the information object address contained within the data itself combine to make the unique address for each data element.

As in DNP, there may be more than one logical or common address for each device in a SCADA system. As for the link level, the address FF or FFFF is defined as a broadcast address. Therefore to send a broadcast message it is necessary to include this address in both the data link and application address fields. Optionally, originator addresses can be carried within the service data unit of the application layer (not shown in Figure 10.8). The information object address is 1 to 3 octets in length, and can be provided either just once within a service data unit of the application layer, or for each separate information object. This allows for efficient transmission of blocks of sequential information.

(3) UCA protocols

UCA is specified and published in the IEEE-SA Technical Report on Utility Communications Architecture- Version 2.0-IEEE-SA IEEE-SA TR 1550-1999. It is designed to apply across all of the SCADA systems for the power, gas, and water facilities, including customer interface, distribution, transmission power plant, control center, corporate information systems, and so on.

The UCA Version 2.0 models, services, and protocols for substation devices are currently being used as the basis for IEC 61850. The initial drafts of IEC 61850 were distributed to IEC member countries for balloting in mid 2000. Many parts of the UCA have also been progressed through the standards process as IEC international standards. The UCA approach to communication between control centers, power plants, and SCADA masters was developed as the Inter-Control Centre Communications Protocol that was later taken up by IEC TC57 working group 7 and standardized as IEC standards 60870-6-503 and 60870-6-802. These standards define methods for synchronizing databases, as well as to perform scheduling, accounting, and other messaging.

It is important to note that UCA is an architecture, rather than a simple protocol. UCA Version 2.0 incorporates a family of basic communications protocols to meet the requirements of a wide range of facility environments. The selection and organization of these protocols has been designed to provide considerable flexibility in choosing the appropriate technology to meet the price and performance criteria of a facility system, while maintaining consistency at the device and data level to reduce integration and vendor product costs. In addition, the UCA includes detailed object models, which define the format, representation, and meaning of utility data. This modeling effort goes far beyond the scope of any other utility communications approach and provides for an unprecedented level of multivendor interoperability. The combination of broad scope and detailed object modeling makes the UCA specifications more complex than a typical protocol document.

UCA differs from most previous utility protocols in its use of object models of devices and device components. These models define common data formats, identifiers, and controls for substation and feeder devices such as measurement unit, switches, voltage regulators, and relays. The models specify standardized behavior for the most common device functions, and allow for significant vendor specialization to allow for future innovation. The models have been developed through an open

process including a broad spectrum of vendor and utility participation. These standardized models allow for multivendor interoperability and ease of integration. Modern protocols (such as those found in UCA) make use of the reduced bandwidth costs and increased processor capabilities in the end devices to carry metadata: standardized names and type information for the most common device information which can be used by applications for online verification of the integration and configuration of databases throughout the utility. Examples of measurement metadata are unit, offset, scale, dead band for reporting, and description. This feature significantly reduces the cost of data integration and data management and reduces down-time due to configuration errors.

10.2.5 SCADA security and reliability

There are two important issues, namely communication security and system reliability, that significantly impact the application of SCADA networks. The first concerns protecting the communication contents and processes in the SCADA networks from attacks by unauthorized persons and even terrorists. The second issue provides an indication how a SCADA system performs for a specific period under given operating conditions.

(1) SCADA communication security

Common vulnerabilities for SCADA networks are thought to arise from three potential sources.

Firstly, many SCADA networks are not isolated or independent networks. In reality, most are linked, or even merged with other computer or industrial control networks. The connections between SCADA and other networks are not necessarily protected by strong access control. Many of the interconnections between SCADA and other networks require compatibilities with different communication standards to move data successfully between two unique systems. Due to the complexity of integrating disparate systems, network engineers often fail to address the added burden of accounting for security risks so access controls are usually minimal.

Secondly, some SCADA networks are designed with insecure network architectures. The architecture is critical in offering the appropriate amount of segmentation between SCADA and other networks, and these interconnections may not be secured by firewall or private network systems consistent with other networks.

Thirdly, many SCADA networks lack real-time monitoring which may lead to vast amounts of data from network security devices overwhelming utility information security resources, which render monitoring attempts futile. When intrusion detection systems are implemented, network security staff can only recognize individual attacks, as opposed to organized patterns of attacks over time.

The most effective communication security strategies for SCADA networks blend regular, periodic security assessments with an ongoing attention to design network architecture and the implementation of real-time monitoring. The following steps highlight major step needed to enhance the SCADA communication security.

(a) STEP 1: regular vulnerability assessments

Many utilities fail to assess the vulnerabilities of their SCADA on a regular basis. In addition to assessing operational systems, corporate networks and customer management systems should also be assessed to reveal unintended gaps in security, including unknown links between public and private networks, and firewall configuration problems.

(b) STEP 2: expert security architecture design

An overwhelming number of security technologies, networking devices, and configuration options are available to utility companies. While firewalls can help protect networks from malicious attacks, improper configuration and/or product selection can seriously hamper the effectiveness of a security policy. In order to minimize risks associated with network architecture design, utilities should be used by information security professionals to ensure that evolving network architectures do not risk information security.

(c) STEP 3: correctly managed security

As companies deploy network security technologies throughout their networks, the proper management and monitoring of these devices is becoming increasingly complex. Many organizations are outsourcing these functions to highly specialized, managed security companies. Managed security services ensure that all security devices are configured properly and fully patched, while monitoring the actual activity on each device to detect malicious activity in real time. They enable corporations or organizations to maintain a real-time security monitoring capability at a relatively low cost, and simultaneously increase the value of existing information security devices by enhancing their performance and capabilities.

(2) SCADA system reliability

For a telemetry system, long-term operational success is dependent upon two factors. The first is reliability, that is, the quality and performance of the system equipment over time. The second factor is the performance of the communication links between the central control and the remote site units.

From a reliability perspective, a telemetry system is a complex engineered system that has many possibilities for failure. These include failure due to components within products; to subsystems; to design flaws; from software applications; to human factors or operating documentation; to environmental conditions; or because redundant equipment fails. This range of possibilities indicate the difficulty of making an analysis that encompasses all these factors, so a reliability analysis is at best an approximation of the worst factors. Reliability prediction techniques used throughout various industries are mostly confined to the mapping of component failures to system failure and do not address these additional factors. Methodologies are currently evolving to model redundancy unit failures, human factor failures and software failures, but there is no model that will provide the level of precision that the existing reliability predictions can achieve based on hardware component failures.

To estimate SCADA reliability rates, an analysis should be conducted of all equipment in the network. For any equipment, the impact of its failure during operation is measured by an index derived from the mean time between failure (MTBF). Table 10.5 lists the orders of worst-case MTBF in hours for typical SCADA network equipment.

Table 10.5 Mean Time Between Failure (MTBF) for SCADA networks

SCADA Equipment	MTBF
Operation station (master station)	30,000 hours
Operator display interface	40,000 hours
Remote terminal units (RTUs)	30,000 hours
Telemetry front ends	120,000 hours

Therefore, it may be necessary to establish redundant items of equipment in the following areas:

(a) the computer display, keyboards, archiving systems, control software and printers, etc. in an operator station at the central control site;
(b) communication media,either landline connections or radio links;
(c) telemetry front-end equipment that connects to the communications medium at the master station and RTU end;
(d) the system power supply, which may be uninterruptible;
(e) all RTUs in field sites.

In addition to setting up redundant items for key system areas, there are three main activities that could improve SCADA reliability:

(a) System design

This includes: reduction in system complexity; duplication to provide fault tolerance; de-rating of stress factors; qualification testing and design reviews; feedback of failure information to provide reliability improvement.

(b) System manufacture

This includes two control activities. The first is the control of materials, tools, methods, changes and environment. The second is the control of work methods and standards.

(c) Field use

This includes: adequate operating and maintenance instructions and training; feedback of field failure information; replacement and spares strategies (for example, early replacement of items with a known wear-out characteristic).

10.3 INDUSTRIAL ETHERNET NETWORK

10.3.1 Introduction

Ethernet as a data communications standard is the basis of the vast majority of office networks today. It supports the transmission of large data packets in soft real-time and high bandwidth, which, coupled with the increased amount of information needed for inter-office communications, i.e. streaming video, video conferencing and presentations, makes this type of network infrastructure ideal for the office environment. In office applications, Ethernet is used to connect PCs to the local area network (LAN), where it can connect to the Internet, printers, mainframes, servers, and other computer networks.

Because of its popularity and its perceived increase in speed in LANs, Ethernet has been widely used in industry for interconnecting the operations of factories or plants. Although Industrial Ethernet is based on the same industry specifications as standard Ethernet technology, implementation of the two solutions is not identical. The primary difference between the two is the type of hardware used. Industrial Ethernet equipment is designed to operate in harsh environments such as grading components, convection cooling, and relay signalling to operate at extreme temperatures and also to resist vibration and shock. Power requirements in industrial environments differ from data networks,

so the equipment runs using 24 volts of direct current power. To maximize network availability, Industrial Ethernet equipment also includes fault-tolerant features such as redundant power supplies. The equipment is also modular, to meet the highly varying requirements of a factory or plant floor.

Industrial Ethernet allows traditional tools and applications to run, but over a much more efficient networking infrastructure. It not only gives manufacturing devices a much faster way to communicate, but also gives the users better connectivity and transparency, enabling connection to the relevant devices without requiring separate gateways. The benefits of Industrial Ethernet networks can be summarized as follows:

(a) Greater bandwidth and speed

Ethernet offers shared bandwidth typically at full-duplex 10 Mbps to 100 Mbps by using switch technologies that can guarantee the throughput to all nodes in the network. This capability allows networks to deliver substantive, actionable information. An Ethernet network, for instance, allows transmission of detailed control information in real-time to a corporation's enterprise resource planning (ERP) system. With enough bandwidth, additional applications can even be added to the network, including those requiring simultaneous data, video, and voice transmission.

(b) Streamlined structure and reduced cost

Traditionally, many manufacturers have maintained separate control networks to support their workshop operations and business activities. Over the years, these networks have been developed in different architectures and protocols. A single Ethernet network eliminates the need to implement, support, and maintain separate systems, so reducing overall network costs and improving information access. A standard Ethernet interface brings to the factory or plant floor the economies of scale enjoyed by hundreds of millions of Ethernet users, lowering costs and increasing the number of potential equipment vendors and products for a particular manufacturing application. In some instances, potential cost reductions can reach an order of magnitude.

(c) Enhanced reliability and virtual determinism

An Industrial Ethernet network is open and transparent, and can support many different protocols simultaneously. Protocols already exist to prioritize data and hence make Industrial Ethernet virtually deterministic, so it is possible to say with complete certainty that an event occurred within a particular time window. Industrial Ethernet employs network switching topologies that reduce message collisions for systems with message cycle times between 10 to 100 milliseconds. To fulfil the more demanding requirements of real-time applications such as machine control, data are transmitted by a specific protocol stack that schedules messages for the purpose of ensuring that all nodes in the network transmit data within a fixed cycle time while simultaneously utilizing such mechanisms as TCP/IP.

(d) QoS (quality of service) and redundancy

For networks which use packet-switched communication, engineering term quality of service (QoS) refers to resource reservation control mechanisms rather than the achieved service quality. For an Industrial Ethernet network, QoS means the ability to apply a higher priority to certain frames. QoS can be implemented on the basis of the port in which the frame arrived to determine the frame priority

(port QoS) or it can use a tag within the frame to determine its priority. These functions are useful in improving determinism.

Redundancy is a popular feature in managed Industrial Ethernet networks. This provides the ability to interconnect switches such that if one interconnecting cable or device fails, another will take over its functionality in the network. Once another link in the network fails, the backup link is automatically enabled, which makes the network uninterrupted.

10.3.2 Industrial Ethernet network system

Traditionally, many types of industry maintain separate networks for workshop operations and business activities due to their different information flows and control requirements. Instead of using architectures composed of multiple separate networks, Industrial Ethernet can allow all systems to run over a single network infrastructure. Figure 10.10 shows a generic Industrial Ethernet system for a corporate IT network, which can be configured at control level and device level.

At the control level, Industrial Ethernet connects control and monitoring devices, including programmable controllers, I/O racks, drives, and HMIs, which were not based upon standard Ethernet and IP in the past, and require a router or a gateway to translate application-specific protocols to Ethernet-based protocols. This translation lets information pass between the control network on the factory floor and the corporate network infrastructure, but has limited functionality and bandwidth, and requires significant effort to keep up to date.

At the device level, the Industrial Ethernet network links the controllers with the corporation's I/O devices, including sensors, transducers, valves, transformers, and other automation and motion equipment, such as robotics, drivers, and actuators. Interconnectivity between these devices was traditionally achieved with a variety of Fieldbuses such as DeviceNet, Profibus, and Modbus. Each Fieldbus has specific power, cable, and communication requirements, depending on the application it supports. This has led to a replication of multiple networks in the same space and the need to have multiple sets of spares, skills, and support programs within the same organization.

(1) Industrial Ethernet hardware

Network components are important for proper functioning of automation and control, and careful consideration should be given to selecting the appropriate device. Most Industrial Ethernet network deployments have used full-duplex Ethernet switches. Switches make it possible for several users to send information over a network at the same time without slowing each other down. In a fully switched network, there are no hubs, so each Ethernet network has a dedicated segment for every node. Because the only devices on each segment are the switch and the node, the switch picks up every transmission before it reaches another node. The switch then forwards the data over to the appropriate segment. In a fully switched network, nodes only communicate with the switch and never directly with each other.

Fully switched Ethernet networks employ either twisted pair or fiber-optic cabling, both of which use separate conductors for sending and receiving data. The use of dedicated communication channels allows the network nodes to transmit to the switch at the same time as the switch transmits to them, eliminating the possibility of collisions. Transmission in both directions can also effectively double the apparent speed of the network when two nodes are exchanging information. For example, if the speed of the network is 10 Mbps, each node can transmit at 10 Mbps at the same time.

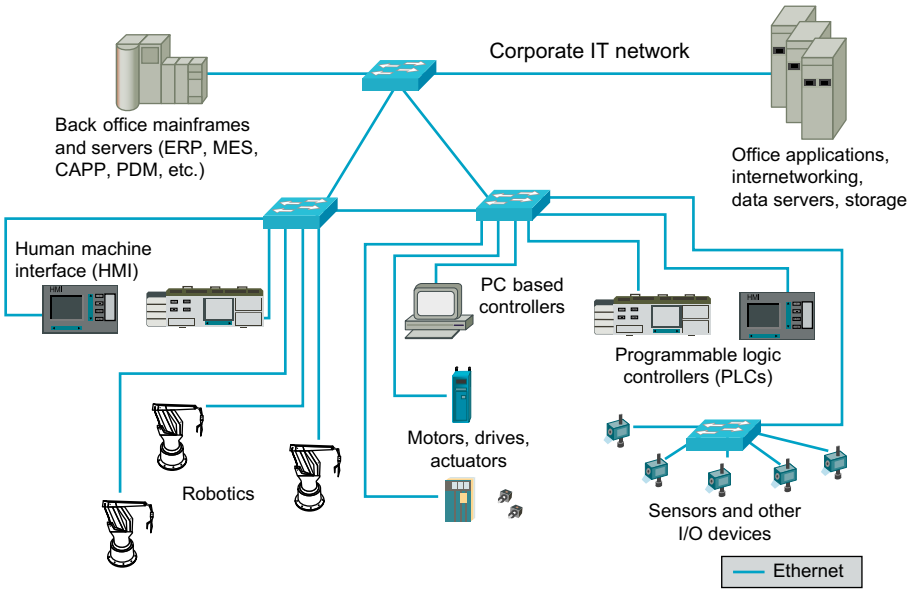


FIGURE 10.10

The generic Industrial Ethernet network; the grey squares and lines indicate Ethernet switches and cables.

(Courtesy of Micrel, Incorporated.)

Real-time Industrial Ethernet switches use a common hardware platform designed to operate with the appropriate protocols. Field-programmable gate array (FPGA) technologies integrate hardware and software technology into Industrial Ethernet switches. When using an Industrial Ethernet module with its related FPGA, it is no longer necessary to design specific hardware for each of the different real-time Ethernet protocols to create a hardware, operating system and TCP/IP suite.

Figure 10.11 shows the components of a real-time Industrial Ethernet module integrated into an Ethernet switch. The Industrial Ethernet module provides all components necessary for the integration of any one of the various real-time Ethernet protocol stacks. Designed for use on slave devices, the Industrial Ethernet module implements a complete Ethernet communication interface between the applications operating on a host CPU and the network. This important feature relieves the host CPU from any communication tasks. The module comprises an FPGA, and further provides sufficient Flash and RAM to run the protocol stack, the TCP/IP stack and the real-time operating system (RTOS). A common application programming interface (API) enables it to connect its application software to the module. The API provides a common software interface for the exchange of process data as well as control and status information between the module and the host controller.

Since the API is largely the same for all the real-time Ethernet protocols, adaptation of the application software to different protocols needs much less effort. For rapid prototyping as well as for low to midrange production volumes, an Industrial Ethernet module with two RJ45 jacks, 3 bi-color LEDs, 8 DIP switches and the connector to the host system is already available for the most important

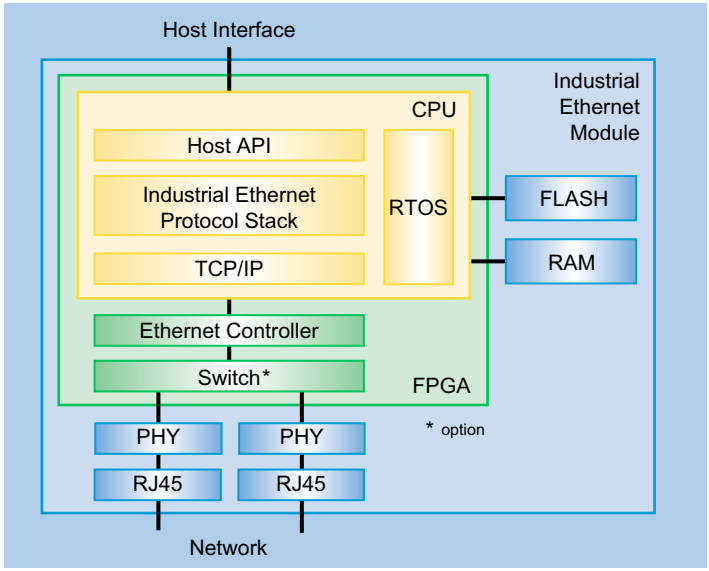


FIGURE 10.11

The components of a real-time Industrial Ethernet module integrated into an Ethernet switch with the FPGA technique.

real-time Ethernet protocols. The number of supported Ethernet protocols will continuously increase, given sufficient industrial requests.

(2) Industrial Ethernet software

Specifically, an Ethernet network normally provides additional software modules that make the network highly functional, manageable, and secure. For industrial environments, this software should include the following:

(a) Port security and access control lists

Provide granular and secure filtering at different network layers, so allowing a network administrator to prevent or allow access to information based on its source, destination, and type of application. Access can be based on physical parameters, IP address, or TCP/UDP port (essentially determining whether the packet is from an application that should be running on the network).

(b) IGMP snooping

Internet group management protocol (IGMP) snooping allows multicast traffic to be easily managed in a switched Ethernet network. Without this feature, the control network could be flooded with multicast traffic. IGMP snooping becomes critical for control applications that use a producer-consumer model in which a stream of data produced by one network node can be used by one or more consumers.

(c) SNMP support

The simple network management protocol (SNMP) forms the basis of almost every major network management system. Intelligent Ethernet devices such as switches must support SNMP, allowing it to interface with existing management systems.

(d) Fast Spanning Tree

The Spanning Tree protocol permits the rapid convergence of a network. If a problem occurs on a network node, a redundant alternative link will automatically come back online by means of this protocol. With Fast Spanning Tree, networks converge very quickly, and a node will become available again in less than 1 second.

10.3.3 Industrial Ethernet network communication

An Industrial Ethernet protocol suite normally consists of the Ethernet protocol suite and the Internet Protocol (IP) suite, which are the standards developed for data communication within automation and control networks. Both the Ethernet and IP protocol suites are associated with a number of technologies and features that support industrial requirements. To accomplish the synchronous data access, Industrial Ethernet equipment must include the intelligence to support features such as multicast control (IGMP Snooping), QoS, and virtual LANs. Other high availability, security, and management functions should also be considered in relation to the specific application.

Industrial Ethernet protocols are significantly different from standard Ethernet protocols. For example, in most automation and control networks, a large percentage of the network traffic is local, where local devices communicate, often by using a multicast model (one sender, many receivers). However, in most office networks the reverse is true, where a large percentage of the network traffic is routed to external locations, such as the Internet, by using a unicast model (one sender, one receiver). Automation and control systems also differ from other applications in their need for determinism and real-time network requirements for rapid and consistent transmission of the data.

Industrial Ethernet is broader than traditional Ethernet technology. In respect of the seven-layer OSI reference model, standard Ethernet technology refers only to layers 1 and 2 (physical and data-link layers), but most Industrial Ethernet solutions also encompass layers 3 and 4 (network and transport layers). This has been done by using IP addressing in layer 3, and Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) in layer 4, in what is referred to as the IP suite. Most controllers (with appropriate network connections) can transfer data from one network type to the other, leveraging existing installations, yet taking advantage of Ethernet. The Fieldbus data structure is applied to layers 5, 6, and 7 of the OSI reference model over Ethernet, IP, and TCP/UDP in the transport layer (layer 4). Please refer to Part 7 of this textbook for a detailed description of the OSI reference model, TCP/IP and UDP.

Industrial Ethernet applies the Ethernet standards developed for data communication to automation and control networks. Figure 10.12 shows an Industrial Ethernet layers model and the protocol module. In such a network, switches usually work at layer 2 (data-link) of the OSI reference model using MAC (media access control) addresses, and deliver a number of important advantages compared to hubs and other LAN devices. In an Industrial Ethernet network, Fieldbus-specific information that is used to control I/O devices and other manufacturing components is embedded into Ethernet frames.

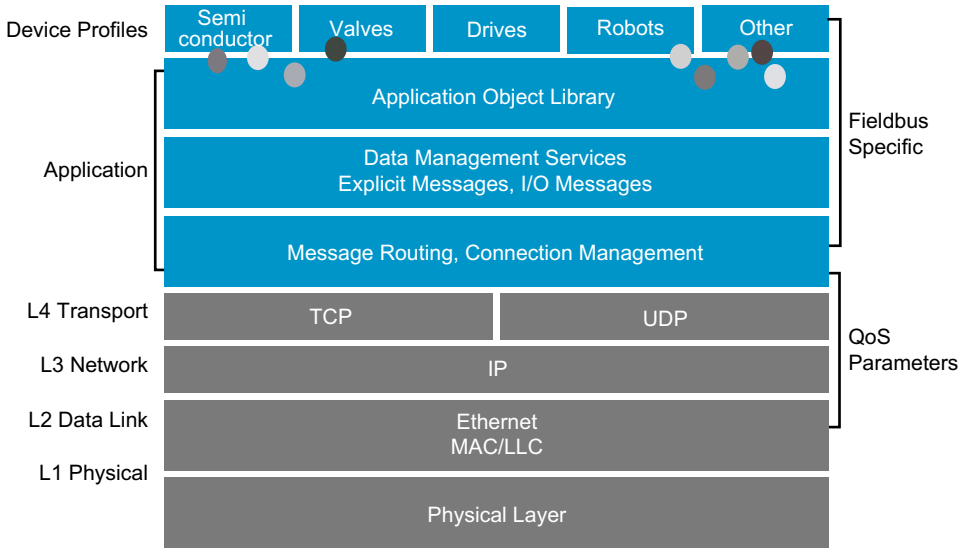


FIGURE 10.12

The Industrial Ethernet network layer model and protocol module.

Because the technology is based on industry rather than proprietary standards, it is more interoperable with other network equipment and networks.

Using IEEE standards-based equipment, industrial organizations can migrate all or part of their automation or control operations to an Ethernet environment at the pace they wish. For example, Common Industrial Protocol (CIP) has implementations based upon Ethernet and the IP protocol suite (Ethernet/IP), DeviceNet, and ControlNet. Below is a list of the current options and a few details about each protocol.

(a) Ethernet/IP

Ethernet/IP was developed by ControlNet International (CI), the Industrial Ethernet Association (IEA) and the Open DeviceNet Vendor Association (ODVA) to be a common application-layer protocol for automation and control systems based on Ethernet technologies such as ControlNet and DeviceNet; it is based on the CIP protocol, which has become a widely accepted standard, for example, in the automobile industry. For hard real-time applications, the extended protocol CIP synchronization was specified. Ethernet/IP is purely a software-based solution.

(b) Profinet

Profinet is an Industrial Ethernet protocol driven by Profibus International. It is designed to be a cross-vendor communication system capable of communicating with different bus systems through a proxy server. There is a real-time solution with capabilities similar to Profibus-DP and an isochronous real-time solution with a jitter of 1 millisecond. The real-time solution is based on software only; for an isochronous real-time ASICs (application-specific integrated circuits) are required.

(c) EtherCAT

EtherCAT is a technology developed with outstanding real-time capabilities. It provides daisy-chain cabling and the Ethernet frame is processed on the fly; each device reads the data addressed to it while the telegram is forwarded to the next device. Telegrams are only delayed by nanoseconds. This excellent performance is related to a specific ASIC that needs to be used in every device.

(d) Modbus TCP

Modbus TCP is an extension of the Modbus family into the Ethernet environment. The Modbus application layer is used on top of Ethernet TCP/IP. It is an open software-based solution without a specific real-time extension. Performance is highly dependent on the design of the Ethernet network and the performance of the processors in the communication interfaces of the devices on that network.

10.3.4 Industrial Ethernet network reliability

Most Industrial Ethernet networks run in real time in order to allow continuous operation with little or no malfunctioning or crashing. They must therefore be reliable. This can be achieved by the use of correct working principles, intelligent networking services, and effective technical solutions. Two important technologies, Power over Ethernet (PoE) and network redundancy, are crucial for an Industrial Ethernet network to operate reliably.

(1) Power over Ethernet

Power over Ethernet or PoE technology describes a system to transmit electrical power, along with data, to remote devices over standard twisted-pair cables in an Ethernet network. In Industrial Ethernet networks, PoE basically "injects" power with data into an Ethernet cable at the source, and "picks" off that power and the data at the destination (note: at the destination, power is just "picked" rather than "eliminated"). The "power injector" and the "power picker" can be external adaptors or can be "embedded" into the devices themselves. Either way, the destination device is completely powered by the Ethernet cable; no extension cord or outlets required. Figure 10.13 illustrates these two techniques as "adapting devices" and "embedded devices" used in PoE. In industry, there are basically three types of PoE techniques.

(a) The first type of PoE uses spare wire pairs. The unshielded twisted pair wiring for Ethernet has eight wires that are twisted in four pairs. Ethernet connects to only two of these four pairs for data, leaving the other two spare. These two spare pairs are used for PoE. All homebrewed and many commercial PoE devices use the spare pairs in unshielded twisted pair wiring, as it is definitely the fastest and easiest way to adapt a non-PoE device to use PoE.

(b) The second type of PoE uses data wires. The IEEE, who standardized Ethernet itself, also standardized PoE in June 2003. The IEEE 802.3af Standard uses the same data pairs as Ethernet, leaving the spare pairs free. This PoE technique adds power to the data pairs using signal transformers, and picks off power at the far end the same way. A comprehensive set of technical standards for power sourcing equipment and powered devices create a safe system protected from shorted wiring, polarity reversal, or accidentally plugging in of non-PoE equipment.

(c) The third type of PoE is a combination of the above two types. The new IEEE 802.3af standard alternatively allows the spare wire pairs to be energized, to be compatible with both types of

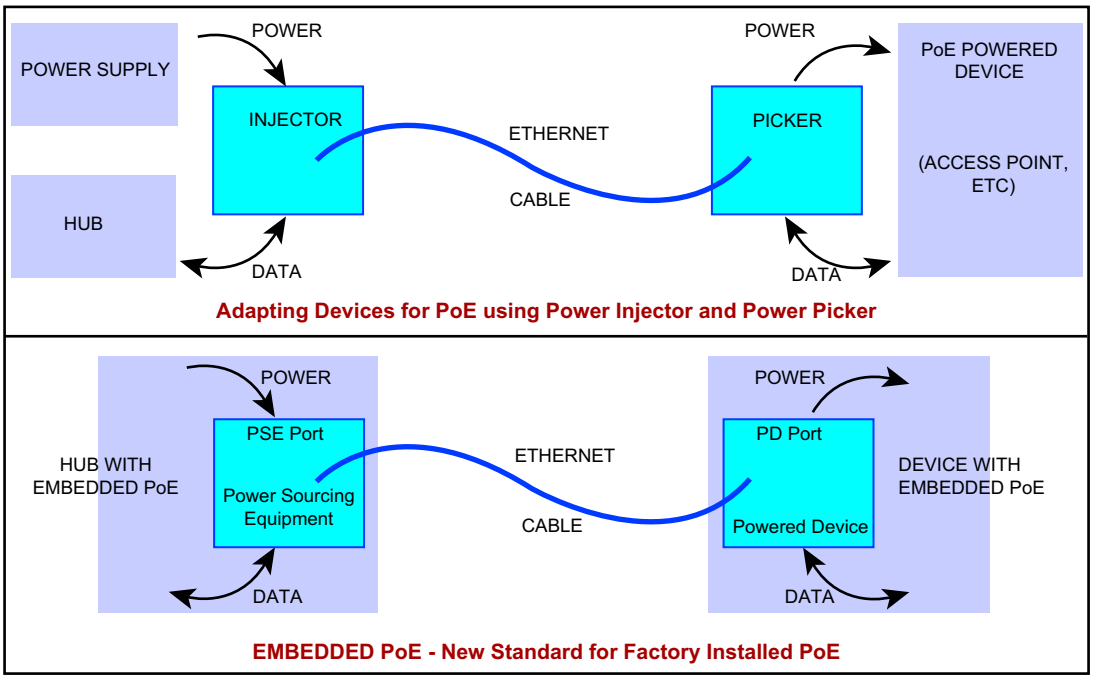


FIGURE 10.13

Two techniques to power an Ethernet cable for PoE.

wiring. This mixed PoE allows brands to migrate to the common standard. The IEEE standard for PoE is mainly used for embedded power, that is, factory built-in PoE. All PoE uses standard unshielded twisted pair wiring for Ethernet cabling and patch panels etc. Also, non-PoE devices are frequently run via PoE by simply running their wall cube power brick output through the Ethernet spare pairs to the device. Any IEEE 802.3af-compatible device will work with any other. If PoE devices are not IEEE 802.3af-compliant, it is best to stick with one brand, or at least with PoE devices known to be compatible with the favorite brand.

As mentioned, the PoE Industrial Standard IEEE 802.3af is publicly accepted worldwide. There are, however, many other proprietary PoE standards, such as Intel, Cisco, and HyperLink etc. This textbook gives only a brief description of IEEE 802.3af due to space limitations.

IEEE 802.3af divides the PoE domain into power sourcing equipment (PSE) such as hubs and routers, and powered devices such as IP phones and wireless access points. Powered devices are classified by the amount of power they consume. Ethernet ports on power sourcing equipment may supply a nominal direct-current power on the data wire pairs or on the spare wire pairs, but not both. A PSE must never send power to a device that does not expect it. PoE is managed by multiple stages of handshake to protect equipment from damage and to manage power budgets. The following briefly discusses how IEEE 802.3af works.

(a) Signature

The power sourcing equipment first probes the device to check whether it is IEEE 802.3af-compliant. Probing with two current limited voltages in a range (say, between 2.7 V and 10 V), the power sourcing equipment looks for a signature impedance of a specified amount (say, 25,000 ohms). The powered device is allowed two diode voltage drops in series with the signature impedance, so two V-I points above the diode drops must be used. Non-PoE devices will usually be below a certain resistance (say, 1,000 or more ohms). If the signature impedance of an IEEE 802.3af device is not seen, the PoE process stops here.

(b) Classification

The power sourcing equipment now forces a classification voltage in a range (say, between 15 V and 20 V) and the power device responds by drawing a specific current to identify its power class according to Table 10.6.

(c) Disconnect

Power sourcing equipment must never send any power to a device that does not expect it, even when connections are changed. Therefore power sourcing equipment is required to remove PoE power when a cable is unplugged, and to reapply power only after the signature and classification phases are correctly repeated. The power sourcing equipment detects the disconnect cable and removes power from it by either of two methods. The direct current disconnect method detects when power device current falls below a given threshold (say, 5 to 10 mA) for a given time (say, 300 to 400 milliseconds). The alternatinge current disconnect superimposes a small alternating current voltage on the power and measures the resulting alternating current, similar to power supply ripple voltage and load ripple current. If the impedance is above a specified resistance (say, 26.25 kohms), power is shut off until a valid signature and classification are established.

Although full compliance with IEEE 802.3af is best done with special chip sets, it is not hard to design your own simple devices to draw PoE power from an IEEE 802.3af-compliant PSE. The easiest way is to implement Class 0 (see Table 10.6) on the sparepairs, with polarity protection, a signature impedance, a voltage threshold and a voltage regulator.

(2) Industrial Ethernet network redundancy

Industrial Ethernet redundancy is the ability of the network to survive a single network cable or device failure in its switch-to-switch links by providing one or more backup data paths when a network cable

Table 10.6 Summary of IEEE 802.3af Power Classifications

Class	Useage	Power Devices' Power (W)	Classification Current (mA)
0	Default	0.44 to 12.95	<5.0
1	Optional	0.44 to 3.84	10.5
2	Optional	3.84 to 6.49	18.5
3	Optional	6.49 to 12.95	28
4	Optional	Reserved*	40

**Class 4 is currently reserved and should not be used.*

or device is at fault. Network redundancy is important if a system or process is highly integrated and if failure in the communication links could result in disastrous consequences such as production loss, poor quality, equipment damage or human injury.

An important index for network performance is the network recovery time; that is, the time it takes to restore the network after a cable or device failure. The faster the recovery time, the better is the redundancy scheme. A few minutes lost in an office environment is merely annoying and inconvenient, but even a few seconds interruption of an industrial communication network can result in significant losses. To maximize system reliability, most proprietary ring networks self-heal in less than 300 milliseconds.

The IEEE has published two protocols that deal specifically with network redundancy; the Spanning Tree Protocol (IEEE 802.1D) and the Rapid Spanning Tree Protocol (IEEE 802.1w). Also available are many proprietary ring technologies and trunking schemes.

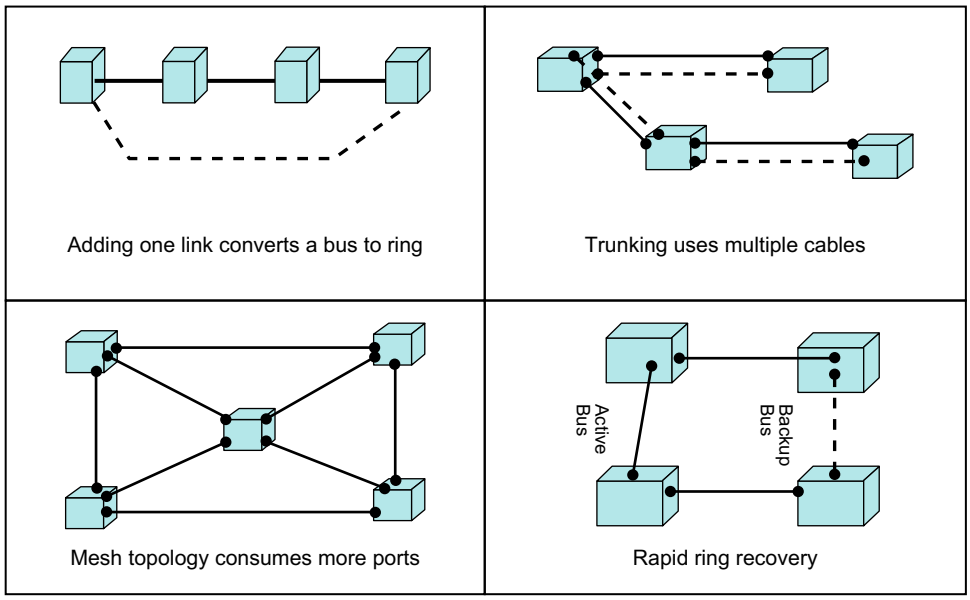
(a) The Spanning Tree Protocol allows networks to be wired in almost any topology, and normally provides network recovery times of between 30 and 60 seconds. Rapid Spanning Tree Protocol is an updated form of the Spanning Tree Protocol and is backward-compatible. This protocol was designed to provide a faster recovery time, generally 1 to 2 seconds. In general, both the Spanning Tree Protocol and the Rapid Spanning Tree Protocol require over 1 second for recovery. To maximize this, both protocol schemes must connect the network as a mesh as shown in Figure 10.14. A mesh requires at least three connections between each switch and neighboring switches.

(b) A ring network is simply a bus with one extra link that connects the last switch to the first switch. In Figure 10.14, the bus connections are shown by the dashed line and the solid line in the "Adding one link converts a bus to a ring" panel is the additional link needed to make a ring. The ring network requires that each switch supports a redundancy protocol, otherwise, a message would travel around the network indefinitely. Ring protocols generally disable one link (backup link) to stop messages from re-circulating around the network. When a link in the ring fails then the backup link is enabled to restore the network, which is shown in Figure 10.14 as "Rapid ring recovery".

Rapid Ring is a proprietary ring network technology that can provide recovery in less than 300 milliseconds. This topology must specify one of its switches as the master to manage the backup link (see Figure 10.14). When a cable break occurs, the master invokes the backup, and each nearby switch alerts the user via relay contact closure, blinking LEDs (light-emitting diodes) and SNMP traps. Once the cable fault is repaired, the nearby switch notifies the master, which then restores normal operation by disabling the backup link.

(c) Trunking allows switches to be interconnected with multiple parallel cables. The more cables between switches, the higher the bandwidth; with selected products, the more levels of redundancy you provide. With two cables between two switches, you have one level of redundancy, and with three cables two levels of redundancy can be achieved and so on. The managed switches from modern controls can provide recovery times of less than 10 milliseconds when using the trunking scheme.

Reliability of Industrial Ethernet networks needs to be implemented at each layer in the OSI reference model, particularly layers 1 and 2: the physical and the data-link layers. Figure 10.15 gives the redundancy schemes for two topologies: start (Figure 10.15 (A)) and ring (Figure 10.15 (B)). In each of these two topologies, the layer 3 (network layer) switch or router is a backup, with one for current working and the other for redundant working.



	Spanning Tree	Rapid Spanning Tree	Ring	Trunking
Recover Time	30 to 60 seconds	1 to 2 seconds	less than 300 milliseconds	less than 10 milliseconds
Flexibility of wiring	High	High	Medium	Low
Protocol	IEEE 802.1D	IEEE 802.1w	Proprietary	Proprietary

FIGURE 10.14

Four types of connection for Ethernet network redundancy.

10.4 INDUSTRIAL ENTERPRISE NETWORKS

10.4.1 Introduction

Automation of production processes, manufacturing facilities and office management in order to improve production efficiency and product quality has been adopted by a multitude of industries world-wide. For this purpose, industrial enterprises need to monitor and control their floor devices and workshop equipment remotely. Many companies have already realized the benefits of networking previously isolated devices and equipment.

This kind of network provides improved and consistent system performance, thus contributing to better process control, product quality and achieving production schedules. Company or factory administrators or managers can also automate their quality control via enterprise or device networking, by ensuring that the critical parameters of their factory systems are monitored, and maintained within appropriate ranges. Monitoring, maintenance and even troubleshooting of production equipment can be conducted remotely in real time. Such a network is defined as an industrial enterprise network or an industrial device network.

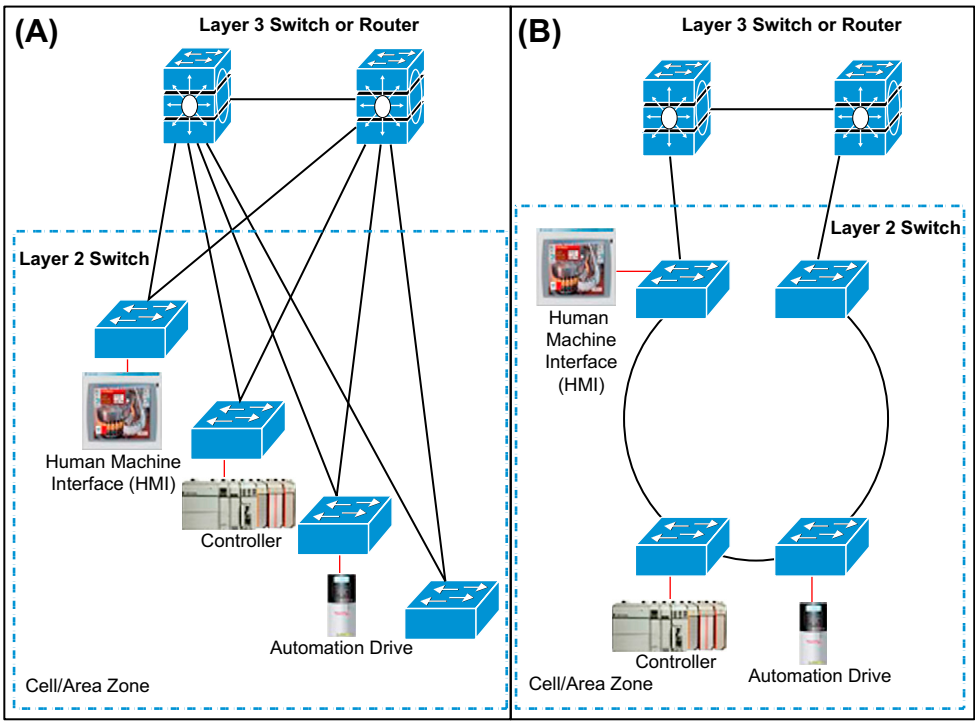


FIGURE 10.15

Two redundancy schemes for: (A) star topology; (B) ring topology.

(Courtesy of Cisco System, Inc.)

System components of industrial enterprise networks must be built to operate in harsh conditions and extreme environments, and must also support standard communication protocols. In some circumstances, these networks also need to resist exposure to electric shock, vibration and physical abuse. Most industrial enterprise networks are required to perform normally and accomplish their designed functions under high temperatures, or in bad weather and other harsh conditions.

To ensure device interoperability, industrial enterprise networking solutions must support protocol models such as ISO/OSI, TCP, IP, Ethernet, Fieldbus, Modbus, HART, RTU and so on. To access the networked devices across the Internet, a built-in web server is needed to allow remote access and management of the attached equipment using a standard web browser. Multiple serial devices can also be cascaded from a single network backbone connection, eliminating the need for expensive switches, hubs, routers, bus, and cables.

Industrial enterprise networking solutions connect enterprise systems to the factory floor and workshop devices without disturbing control networks or needing dedicated wiring for remote working. An industrial networking solution should therefore enable connection of virtually any piece of factory equipment to a network or to the Internet access, management, control, and repair, and even automatic data capture.

Such a network will also allow engineers and managers to leverage their existing network wiring and corporate IP networks. To collect information from this network, it must be able to be used directly with two systems typically found in most industrial operations: a human machine interface (HMI) and a supervisory control and data acquisition (SCADA) system.

In the past decades, enterprises have enabled remote monitoring and managing in real time by placing their equipment on an existing local area network (LAN). Indeed, LAN networking provides a solution which satisfies all the requirements of industrial enterprise networks. Therefore, LAN has become a dedicated architecture for implementing industrial enterprise networks. The following are two examples of the applications of LAN for industrial enterprise networks.

(a) Industrial enterprise networking example: Rockwell Automation

Rockwell Automation is one of the leading global providers of industrial automation power, control, and information solutions that help customers meet their manufacturing objectives. One of the company's leading brands is Allen-Bradley, a manufacturer of automation controls and a provider of engineering services. Allen-Bradley control solutions have set a high standard in industrial automation, helping the industry apply programmable logic controller (PLC) technology over the past decades.

Rockwell's customers needed to remotely access and manage their PLCs when only a serial interface was available. To meet their needs, Rockwell designed an industrial device networking server to provide Ethernet/IP connectivity. Their server provided a gateway from their controllers' serial port to an Ethernet network. This allowed their customers to upload and download programs, communicate between controllers, and generate email messages via SMTP (simple mail transport protocol). Rockwell Automation has enhanced the capabilities of their PLCs and allowed their customers to remotely access their controllers from anywhere in the world.

(b) Industrial enterprise networking example: Texas Instruments

Texas Instruments is one of the world leaders in digital signal processing and analog technologies that drive semiconductor engines. This company needed secure remote access to all its process control equipment at the company's support center, while keeping costs and wiring to a minimum. The company needed to measure and read the concentration of contaminants in water samples. Before industrial enterprise networking was deployed, the process used was complex, involving the transfer of a signal from a water analyser to a PLC, and then to an HMI, where reading of the measurement was often flawed. While the company's facility control center operated process control equipment on a legacy network, independent of their LAN, they needed to network-enable all of the process control equipment at the support center, which would have required 1,500 feet of wiring and conduit spanning multiple buildings for added expense and time.

By implementing industrial enterprise networking with a LAN solution on multiple key pieces of equipment such as airflow, water and gas detectors at its fabrication facility, its support center could remotely monitor and control critical elements of the fabrication plant (airflow, water treatment and gas detection) in an adjacent facility. By integrating the industrial device networking solution, all of its equipment in the support center is now Ethernet-enabled, allowing more than 500 PCs in the center to have access in real time to information as it is generated by the process control equipment. As a result, it is no longer necessary for a technician to patrol the floor of the plant to monitor each device individually and response time is significantly improved whenever a failure is detected.

These two examples demonstrate to some extent how industrial enterprise networks can revolutionize the world of industrial production and manufacturing. The adoption of industrial enterprise networking is increasing as local area network (LAN) or wide area network (WAN) architecture becomes more available. Since there seems to be no substantial difference between the LAN and the WAN for industrial automation and controls, this textbook uses LAN as a unified definition of LAN and WAN. LAN architecture has developed significantly over the past decades, generating two important forms of LAN; the virtual local area network (VLAN) and the wireless local area network (WLAN).

Virtual LANs (VLANs) have developed into an integral feature of the switched LAN solutions of every major LAN equipment vendor. Although end-user enthusiasm for VLAN implementation has yet to take off, most organizations have begun to look for vendors that have a well-articulated VLAN strategy, as well as VLAN functionality built into their products. One of the reasons for the attention is the rapid deployment of LAN switching that began in 1995.

Wireless technology has revolutionized how computer users access information. And this revolution continues in the world of industry. Wireless device networking is the best alternative when it is impractical or cost-prohibitive to run cabling to connect factory equipment to a LAN or the Internet. Wireless connection reduces the need for expensive wiring, which can be upto two-thirds of the total cost of an installation in an industrial enterprise.

10.4.2 Local area network (wired LAN)

In industrial applications, the local area network (LAN) is a data communications network connecting computers and other programmable intelligent devices such as terminals, controllers, field equipment, and office facilities within a building, a workshop, a factory or other geographically limited areas. These devices can be connected via wired cables or wireless links.

The wide area network, often referred to as a WAN, is also a data communication network but it connects local computer networks into a larger working network that may cover city, national or international locations. Wide area networks can also be connected via wired cables or wireless links.

Linking one local area network with another is often desirable, especially for businesses that operate a number of facilities. To scale up from the local area network to the wide area network is most easily accomplished by using telephony. Essentially, fiber-optics are used to create the link between networks of different facilities. This often uses standard phone lines, or employs PSTN (public switched telephone network) technology. During the 1990s, a third option, that of ISDN (integrated services digital network) solutions for creation of a wide area network, gained a great deal of popularity, mainly it then become more cost-effective to extend the network beyond national boundaries.

(1) LAN architectures

There are three logical network models for both wired and wireless local area networks: peer-to-peer network, client-server network and distributed-services networks.

A peer-to-peer network is composed of two or more self-sufficient computers or stations. Each handles all functions; logging in, storage, providing a user interface, etc. The computers or stations on a peer-to-peer network can communicate, but do not need the resources or services available from the other computers or stations on the network. Peer-to-peer is the opposite of the client-server logical network model. UNIX servers running as stand-alone systems form a peer-to-peer network. In Figure 10.16, there are two workgroup panels connecting to their respective shared-media hub in the

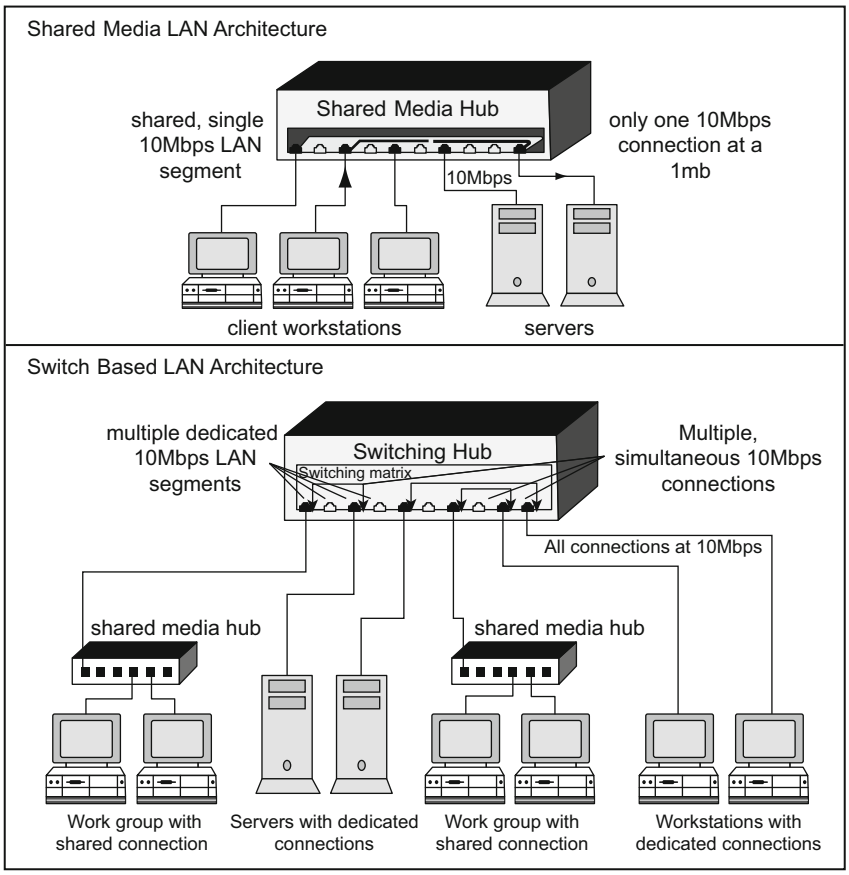


FIGURE 10.16

Wired LAN architectures: shared-media LAN architecture (upper); switch-based LAN architecture (lower).

lower box for "Switch-Based LAN Architecture". The peer-to-peer logical model exists in each of the two workgroups.

A client-server network is composed of a server and one or more clients. The server provides a service that the client needs. Clients connect to the server across the network in order to access the service. A server can be a piece of software running on a computer or station; or the computer or station itself. Most computer networks control logins on all machines from a centralized logon server. When you sit down to a computer or station and type in your username and password, they are sent to the logon server across the network. In Figure 10.16, the upper box should be a client-server network because two server stations and more than three client stations are physically connected by one shared-media hub. However, in some cases, both server and client can be a software package existing in a computer or network.

A distributed computer network provides services to client computers or stations, but not from a centralized server. The services run on more than one computer or station, and some or all of the

functions provided by the service are provided by more than one server. The simplest example of a distributed service is domain name service (DNS), which performs the function of turning human-understandable names into numerical IP addresses. Whenever a web page is accessed, the client computer sends a DNS request to the local DNS server. That local server will then go to a remote server on the Internet called a DNS root server to begin the lookup process. This root server will then direct the local DNS server to the owner of the domain name the website is a part of. Thus, there are at least three DNS servers involved in the process of finding and providing the IP address of the required website.

(2) LAN topologies

The components in a LAN can be connected in several ways, called LAN topologies. Figure 10.17 gives the four basic LAN topologies.

(a) Star

All stations are connected by cable (or wirelessly) to a central point, such as hub or a switch. If the central node is operating in a broadcast paradigm such as a hub, transmission of a frame from one station to the node is repeated on all of the outgoing links. In this case, although the arrangement is physically a star, it is logically a bus. If the central node acts as switch, an incoming frame is processed

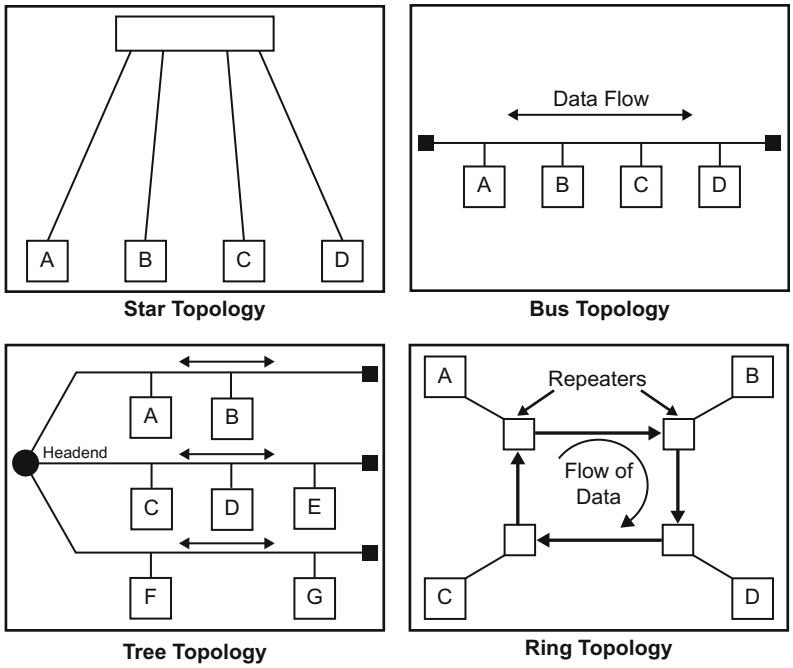


FIGURE 10.17

Wired LAN topologies: star, bus, tree and ring.

at the node and then retransmitted on an outgoing link to the destination station. Ethernet protocols (IEEE 802.3 standard) are often used in star topology LAN.

(b) Ring

All nodes on the LAN are connected in a loop and their network interface cards (NICs) work as repeaters. There is no start or end point. Each node will repeat any signal on the network regardless of its destination. The destination station recognizes its address and copies the frame into a local buffer as it goes by. The frame continues to circulate until it returns to the source station, where it is removed. Token ring (IEEE 802.5 standard) is the most popular ring topology protocol. FDDI (IEEE 802.6 standard) is another protocol used in the ring topology, which is based on the token ring.

(c) Bus

All nodes on the LAN are connected by one linear cable, which is called the shared medium. Every node on this cable segment sees transmissions from every other station on the same segment. At each end of the bus is a terminator which absorbs any signal, removing it from the bus. This medium cable is the single point of failure. Ethernet (IEEE 802.3 standard) is the protocol used for this type of LAN.

(d) Tree

The tree topology is a logical extension of bus topology. The transmission medium is a branching cable with no closed loops. The tree layout begins at a point called the head-end, where one or more cables start, and each of these may have branches. The branches in turn may have additional branches to allow quite complex layouts.

The wired LAN segment can be run on a variety of connection media, including fiber-optics, twisted-pair, and cable. The LAN segments can be joined into a larger network. The following briefly introduces the LAN technologies for joining the LAN segments. In applications, any LAN network will need one or more than one of these technologies.

(a) Bridges

A bridge is a programmable intelligent device that connects physically separate LAN segments into one logical LAN segment. There are four categories of bridge: transparent, source routing, encapsulating, and translating. Transparent bridges are used for Ethernet, source routing bridges for token ring networks. Encapsulating bridges connect two segments of the same media (such as token ring to token ring) over a medium. The receiving bridge removes the envelope, checks the destination, and sends the frame to the destination device. Translating bridges are used to connect different types of network media such as Ethernet and FDDI (fiber-optic distributed data interface). FDDI is a set of protocols that uses a modified form of the token-passing method over fiber-optic cable.

(b) Routers

LAN segments joined by a router are physically and logically separate networks. In contrast to a bridge, when multiple network segments are joined by a router they maintain their separate logical identities (network address space), but constitute an inter-network. Routers specify the destination and route for each packet, and can be used to direct packets and interconnect a variety of network architectures. A major difference between a bridge and a router is that the bridge distinguishes packets by source and destination address, whereas a router distinguishes packets by protocol type. Routers provide the

interfaces to WAN, such as frame relay and packet switching services. Some new bridge products have additional router capabilities. Routers can also be used to limit access to a network by the type of application (for example, allowing electronic mail to pass, but not file transfer traffic). This capability provides a measure of security for the network, and is used extensively when creating firewalls.

(c) Switches

A switch, like a hub, connects nodes to each other. However, while a hub requires each node to share the bandwidth (i.e., the amount of simultaneous data traffic the network can support), a switch allows each node to use the full bandwidth. In a fully switched network, each node is connected to a dedicated segment of the network, which in turn is connected to a switch. Each switch supports multiple dedicated segments. When a node sends a signal, the switch picks it up and sends it through the appropriate segment to the receiving node. Ethernet protocol in a fully switched environment does not require collision detection because the switches can send and receive data simultaneously, thus eliminating collisions. Adding switches to the network is one way to reduce collisions.

(d) Hub

A hub is a common connection point for devices in a network, commonly used to connect segments of a LAN. A hub contains multiple ports. When a packet arrives at one port, it is copied to the other ports so that all segments of the LAN can see all packets. There are three types of hub; passive hub, intelligent hub, and switched hub. A passive hub serves simply as a conduit for data, enabling them to go from one device (or segment) to another. So-called intelligent hubs include additional features that enable an administrator to monitor the traffic passing through the hub and to configure each port. Intelligent hubs are also called manageable hubs. A switching hub actually reads the destination address of each packet and then forwards the packet to the correct port.

(e) Repeater

Repeaters are used in the transmission of information throughout a LAN, commonly used to increase the number of available ports. They are simple devices for broadcasting data packets originating at one port to all other ports. Multiple end-stations or computers connect to repeaters with coaxial cables, twisted-pair wiring, or fiber-optics. A typical repeater comprises a single integrated circuit chip. Each chip has a limited number of ports. A network repeater typically has four, eight, or sixteen ports.

(3) LAN protocols

In 1983, the International Standards Organization (ISO) developed a network model called the Open Systems Interconnection (OSI) Reference Model, which defined a framework of computer communications. The ISO/OSI Reference Model (ISO/OSI model) has seven layers: physical, data-link, network, transport, session, presentation, and application layers.

The physical layer physically transmits signals across a communication medium. The data-link layer transforms a stream of raw bits (0s and 1s) from the physical layer into an error-free data frame for the network layer. This then controls the operation of a packet transmitted from one network to another. The transport layer splits data from the session layer into smaller packets for delivery on the network layer and ensures that the packets arrive correctly at their destination. The session layer establishes and manages sessions, conversions, or dialogues between two computers. The presentation layer manages the syntax and semantics of the information transmitted between two computers. The

application layer, the highest layer, contains a variety of commonly used protocols, such as file transfer, virtual terminal, and email.

The Institute of Electrical and Electronic Engineers (IEEE) have developed a set of LAN standards, known as IEEE Project 802, which the ISO accepted as international standards. The IEEE LAN standards addressed only the lowest two layers; the physical and data-link layers, of the ISO/OSI model. In a LAN, TCP/IP protocol is normally used in the network and transport layer, as shown in Figure 10.12.

Figure 10.18 shows the correspondences between the ISO/OSI reference model and the LAN networks with repeater, bridge, and router, respectively. The figure indicates that the repeater only performs the physical layer function; the bridge performs the functions specified by both physical and data-link layers; the router performs the functions of physical, data-link, and network layers.

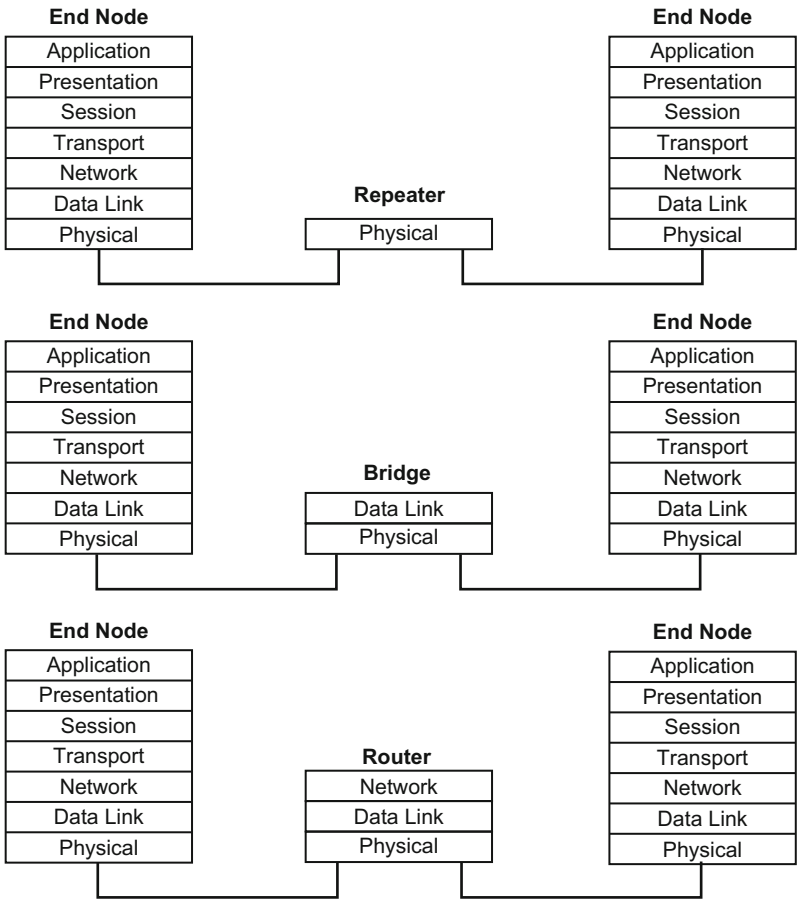


FIGURE 10.18

The correspondences between the OSI reference model and LAN protocols for the LAN networks with repeater, bridge, and router, respectively.

Table 10.7 Local Area Network (LAN) Protocols and IEEE Standards

Network Types	IEEE Standards
Wired LAN	Token Ring: IEEE 802.5 LAN protocol
	FDDI: Fiber Distributed Data Interface, and IEEE 802.6
	LLC: Logic Link Control (IEEE 802.2)
	SNAP: SubNetwork Access Protocol
	STP: Spanning Tree Protocol (IEEE 802.1D)
	IEEE 802.1p: LAN Layer 2 QoS/CoS Protocol
Wireless LAN	Wireless LAN in IEEE 802.11, IEEE 802,11a, IEEE 802.11b, IEEE 802.11g and IEEE 802.11n
	IEEE 802.11i: WLAN Security Standards
	IEEE 802.1X: WLAN Authentication & Key Management
	IEEE 802.15: Bluetooth for Wireless Personal Area Network (WPAN)
Virtual LAN	IEEE 802.1Q: Virtual LAN Bridging Switching Protocol
	IEEE 802.1P: Generic Attribute Registration Protocol
	IEEE 802.1P: Multicast Registration Protocol
	IEEE 802.1P, IEEE 802.1Q: VLAN Registration Protocol
	VTP: VLAN Trunking Protocol
Ethernet	Ethernet LAN protocols as defined in IEEE 802.3 suite
	Fast Ethernet: Ethernet LAN at data rate 100Mbps (IEEE 802.3u)
	Gigabit Ethernet: Ethernet at data rate 1000Mbps (IEEE 802.3z, IEEE 802.3ab)
	10Gigabit Ethernet: Ethernet at data rate 10 Gbps (IEEE 802.3ae)

Table 10.7 lists the IEEE Standards for wired, wireless, virtual, and Ethernet LAN protocols.

(4) LAN media-access methods

LANs use CSMA/CD technology to deal with devices contending for the network media. Normally CSMA/CD networks are half-duplex, meaning that a device sending information on cannot receive at the same time i.e., while that device is talking, it is incapable of also listening for other traffic. This is much like a walkie-talkie: when one person wants to talk, he presses the transmit button and begins speaking. While he is talking, no one else on the same frequency can talk. When the sending person has finished, he releases the transmit button and the frequency is available to others.

When a device has data to send, it first listens to see whether any other device is currently using the network. If not, it starts sending data. After finishing its transmission, it listens again to see whether a collision has occurred; if two devices send data simultaneously. When a collision happens, each device waits a random length of time before resending its data. In most cases, a collision will not occur again. Because of this type of network contention, the busier a network becomes, the more collisions occur, which is why the performance of Ethernet degrades rapidly as the number of devices on it increases.

In token-passing networks such as token ring and FDDI, a special network frame called a token is passed around the network from device to device. When a device has data to send, it must wait until it

has the token and then sends its data. When the data transmission is complete, the token is released so that other devices may use the network media. The main advantage of token-passing networks is that they are deterministic. In other words, it is easy to calculate the maximum time that will pass before a device has the opportunity to send data. Token-passing networks are therefore popular in some real-time environments such as factories, where machinery must be capable of communicating at a determinable interval.

For CSMA/CD networks, switches segment the network into multiple collision domains. This reduces the number of devices per network segment that must contend for the media. By creating smaller collision domains, the performance of a network can be increased significantly without requiring addressing changes. Although CSMA/CD networks are normally half-duplex, when switches are introduced full-duplex operation is possible. When a network device is attached directly to the port of a network switch, the two devices may be capable of operating in full-duplex mode. A 100-Mbps Ethernet segment is capable of transmitting 200 Mbps of data, but only 100 Mbps can travel in one direction at a time. Because most data connections are asymmetric (with more data travelling in one direction than the other), the gain is not as great as many claim. However, full-duplex operation does increase the throughput of most applications because the network medium is no longer shared. Two devices on a full-duplex connection can send data as soon as they are ready. Token-passing networks such as token ring can also benefit from network switches. In large networks, the delay between turns to transmit may be significant as the token is passed around the network.

(5) LAN transmission methods

LAN data transmissions fall into three classifications: unicast, multicast, and broadcast. In each type of transmission, a single packet is sent to one or more nodes.

In a unicast transmission, a single packet is sent from the source to a destination on a network. First, the source node addresses the packet with the address of the destination node. The package is then sent onto the network, and finally, the network passes the packet to its destination.

A multicast transmission consists of a single data packet that is copied and sent to a specific subset of nodes on the network. First, the source node addresses the packet with a multicast address. The packet is then sent into the network, which makes copies of the packet and sends a copy to each node that is part of the multicast address.

A broadcast transmission consists of a single data packet that is copied and sent to all nodes on the network. In these types of transmissions, the source node addresses the packet by using the broadcast address. The packet is then sent on to the network, which makes copies of the packet and sends a copy to every node on the network.

10.4.3 Virtual local area network (VLAN)

Virtual LAN, VLAN, refers to a group of networked devices existing on a number of different LANs or different LAN segments that are logically configured in such a way that they can communicate as if they belonged to the same LAN, or were attached to the same linkage. It is important that a VLAN is based on logical instead of physical connections. VLAN is analogous to a group of end-stations, perhaps on multiple physical LAN segments, that are not constrained by their physical location and can communicate as if they were in the same LAN or on a common segment.

From a technical perspective, VLAN provides two key features; it uses broadcasting transmission methods, but replaces routers and shared media devices such as hub with LAN switches. With the rapid decrease in the price of Ethernet and switch-on per port, many organizations are now adopting VLAN implementation.

One of the objectives of VLAN is to establish a virtual workgroup model. This means that members of the same department or section can all appear to share the same LAN, with most of the network traffic staying within the same VLAN broadcast domain. Someone moving to a new physical location but remaining in the same department could remain connected without having workstations reconfigured. Conversely, a user would not have to change his or her physical location when changing departments the network manager would simply change the user's VLAN membership information.

The ability of VLANs to create firewalls can also satisfy more stringent security requirements and thus replace much of the functionality of routers in this area. This is because the only broadcast traffic on a single-user segment would be from that user's VLAN (that is, traffic intended for that user). Conversely, it would also be impossible to "listen" to broadcast or unicast traffic not intended for that user (even by putting the workstation's network adapter in promiscuous mode), because such traffic does not physically traverse that segment.

In practice, industrial organizations migrate from existing enterprise networks with LAN architectures to VLAN architectures in ways that are based on their business models and industrial operations. Two types of VLAN approaches are applicable to industrial organizations:

(a) Infrastructural VLANs

An infrastructural approach to VLANs is based on the functional groupings (that is, the departments, factories. divisions, sections, etc.) that make up the organization. Each functional group, such as accounting, sales, personnel, and engineering, is assigned to its own uniquely defined VLAN. In this approach, VLAN overlap occurs at network resources that must be shared by multiple workgroups. These resources are normally servers, but could also include printers, terminals, routers providing LAN access, computers or workstations functioning as gateways, etc. This approach is the easiest to deploy with existing technology and fits more easily into existing networks. Moreover, this approach does not require network administrators to alter how they view the network, and has lower deployment costs. An example of such a deployment can be seen in the upper panel of Figure 10.19, where the e-mail server is a member of all of the departments' VLANs, while the accounting database server is only a member of the accounting VLAN.

(b) Service-based VLANs

A service-based approach to VLAN implementation is based, not on functional infrastructures, but on individual user access to the network resources existing in the organization. In this model, each network resource corresponds to a server or service on the network. These servers do not belong to multiple VLANs, which is the main difference from the infrastructure model. For example, in a given organization, all users would belong to the e-mail server's VLAN, but only a specified group such as the accounting department plus executives would be members of the accounting database server's VLAN. By its nature, the service-based approach creates a much more complex set of VLAN membership relationships to be managed. Figure 10.19 depicts a service-based VLAN model in its lower panel.

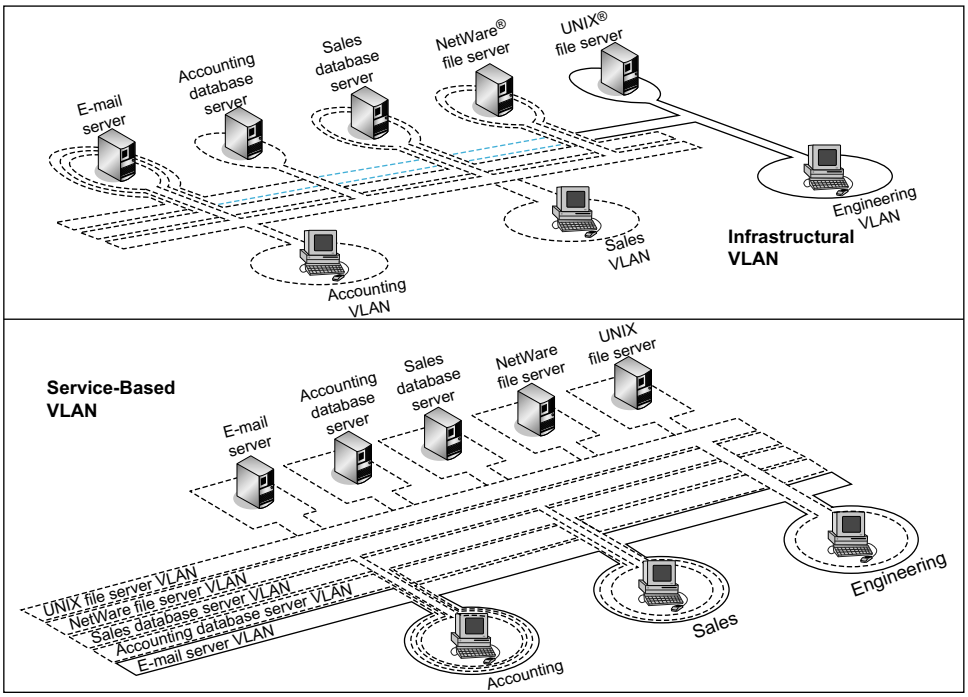


FIGURE 10.19

Two types of virtual LAN (VLAN) system: upper panel is the infrastructural VLAN; lower panel is the service-based VLAN.

(Courtesy of DeciSys, Inc.)

What steps are necessary before applying VLANs to an enterprise network? Initially, VLANs should be seen as a solution to at least one of two problems: containment of broadcast traffic to minimize dependence on routers; and reduction in the cost of network moves and changes.

Industrial enterprises which are replacing routers with switches, and may face broadcast traffic containment issues, another element of the network architecture should be considered: the degree to which the network has evolved toward a single user or port switched LAN architecture. If the majority of users are still on shared LAN segments, the ability of VLANs to contain broadcasts is greatly reduced. If multiple users belonged to different VLANs on the same shared LAN segment, that LAN segment would receive broadcasts from each VLAN, which defeats the goal of broadcast containment.

Having determined that VLANs need to be a part of network planning in the immediate future, server access, server location, and application utilization must all be thoroughly analyzed to determine the nature of traffic flow in the network. This analysis should identify where VLAN broadcast domains should be deployed, what role the ATM (asynchronous transmission model) needs to play, and where the routing functions should be placed.

(1) Defining VLAN

The implementation of a VLAN is by logical configuration rather than by physical connection. Virtual LAN has four types of configuration method; port-based, MAC-based (MAC means media-access control; the MAC driver is one sublayer of the data-link layer in the ISO/OSI Reference Model), protocol-based, and ATM-based. These four configurations are oriented to the respective layer protocols of layer 1 to layer 4 as defined in the ISO/OSI Reference Model. The port-based configuration is oriented to the physical layer (layer 1) protocol, the MAC-based is oriented to the data-link layer (layer 2) protocols, the protocol-based is oriented to the network layer (layer 3) protocols, and the ATM-based is oriented to the transport layer (layer 4) protocols.

(a) Port-based VLAN configuration (layer 1, physical layer oriented)

Many VLANs are initially implemented by dividing membership into groups of switch ports. In this method, each physical switch port is configured with an access list that specifies its membership of a set of VLANs. For example, ports 1, 2, 3, 7, and 8 on a switch make up VLAN A, while ports 4, 5, and 6 make up VLAN B. Furthermore, in most initial implementations, VLANs can only be supported on a single switch. Second-generation implementations support VLANs that span multiple switches (for example, ports 1 and 2 of switch 1 and ports 4, 5, 6, and 7 of switch 2 make up VLAN A; while ports 3, 4, 5, 6, 7, and 8 of switch 1 combined with ports 1, 2, 3, and 8 of switch 2 make up VLAN B). This scenario is depicted in Figure 10.20. Port grouping is still the most common method of defining VLAN membership, and configuration is fairly straightforward. Defining VLANs purely by port group does not allow multiple VLANs to include the same physical segment (or switch port). However, the primary limitation of defining VLANs by port is that the network manager must reconfigure VLAN membership when a user moves from one port to another.

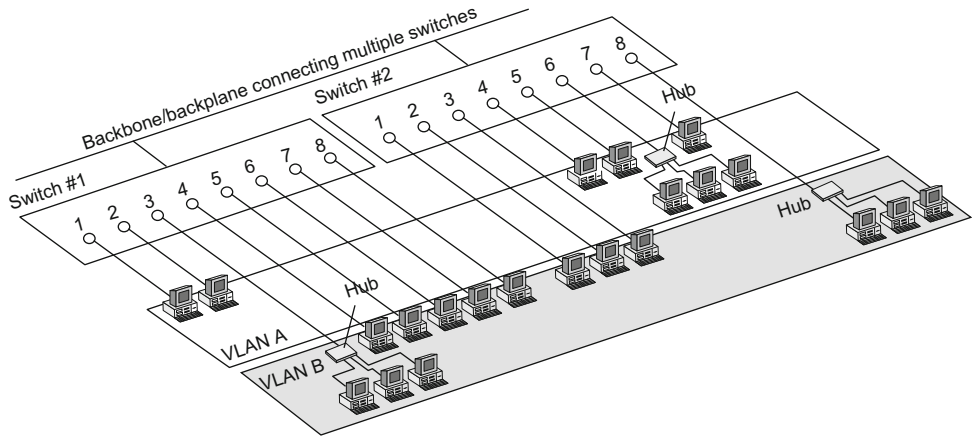


FIGURE 10.20

The VLANs defined by port-based configuration.

(Courtesy of DeciSys, Inc.)

(b) MAC-based VLAN configuration (layer 2, data-link layer oriented)

In this method, a switch rather than the switch ports is configured with an access list mapping individual MAC-sublayer addresses to VLAN membership. Because these addresses are hard-wired into the network interface card (NIC) of a computer or end-station, VLAN membership is configured on the basis of MAC-sublayer address. With this configuration method, a workstation can move to a different physical location without reconfiguration, but all users must be initially configured to be in at least one VLAN. In shared media environments, performance will degrade as members of different VLANs share a single switch port.

(c) Protocol-based VLAN configuration (layer 3, network layer oriented)

VLAN configurations based on layer 3 information take into account protocol types. In applying this method, a switch is configured with a list mapping layer 3 protocol types to VLAN membership; therefore filtering IP traffic from nearby computers or end-stations which use a particular protocol, such as IPX. Multiple protocols or network-layer addresses must be supported (for example, subnet addresses for TCP/IP networks). Although these VLANs are based on layer 3 information, this does not constitute a "routing" function and should not be confused with network-layer routing. Having made the distinction between VLANs based on layer 3's information and routing, connectivity within any given VLAN system is still seen as a bridged topology.

(d) ATM-based VLAN configuration (layer 4, transport layer oriented)

This approach uses the LAN Emulation protocol to map Ethernet packets onto ATM cells, and deliver them to their destination by converting an Ethernet's MAC-sublayer address into an ATM address. Figure 10.21 and the following briefly illustrates how LAN Emulation operates:

1. The LAN switch receives a frame (or message) from an Ethernet-connected end-station. This frame is destined for another Ethernet end-station across the ATM backbone.
2. The LAN-emulation client (which in this case resides in the LAN switch) sends a MAC-to-ATM address resolution request to the LAN-emulation server (which in this case resides in an ATM switch).
3. The LAN-emulation server sends a multicast to all other LAN-emulation clients in the network.
4. However, only the LAN-emulation client that has the destination (MAC) address in its tables responds to the LAN-emulation server.
5. The LAN-emulation server then broadcasts this response to all other LAN-emulation clients.
6. The original LAN-emulation client recognizes this response, learns the ATM address of the destination switch (note: this assumes a connection-oriented ATM), and sets up a switched virtual circuit to transport the frame via ATM cells, which governs segmentation and reassembly.

Another issue crucial to VLAN deployment is the degree to which the VLAN configuration is automated. There are three primary levels of automation in VLAN configuration: manual, semi-automated, and fully automated. To a certain extent, this degree of configuration automation is correlated to how VLANs are defined.

(2) Implementing VLAN

Figure 10.22 shows a typical virtual LAN system architecture. In this figure, the VLAN system is abstracted into the three backbone layers: core, distribution, and access. The core layer consists of

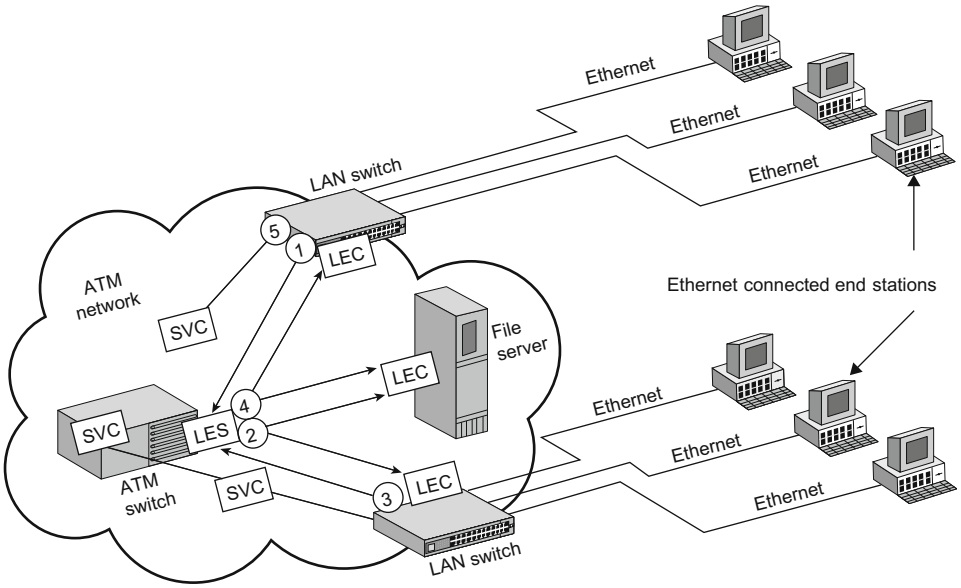


FIGURE 10.21

An ATM-VLAN network deployed by LAN emulation (LEC, LAN emulation client; LES, LAN emulation server; SVC, switch virtual circuit).

high-speed switches which play the role of the system root controllers. The next layer is the distribution layer, located between the core switches and all the joined LAN networks, which also consists of high-speed switches connecting to the core and LAN branches. The layer consisting of all the joined branches is named the access layer, in which all the LANs joined to this VLAN system are connected to the switches in the distribution layer.

To date, three methods have been implemented for inter-switch communication of VLAN systems: table maintenance via signaling, frame-tagging, and time-division multiplexing (TDM).

Table maintenance via signaling operates as follows: when an end-station broadcasts its first frame, the switch resolves the end-station's MAC address or attached port with its VLAN membership by consulting cached address tables. This membership information is then broadcast continuously to all other switches. As VLAN membership changes, these address tables are manually updated by a system administrator at a management console. As the network expands and switches are added, constant signaling is necessary to update the cached address tables of each switch.

In the frame-tagging approach, a header is usually inserted into each frame on inter-switch trunks to uniquely identify which VLAN a particular MAC sublayer frame belongs to.

The third, and least utilized, method is time-division multiplexing (TDM). TDM works in the same way on the inter-switch backbone to support VLANs as it does in the LAN environment to support multiple traffic types; here, channels are reserved for each VLAN.

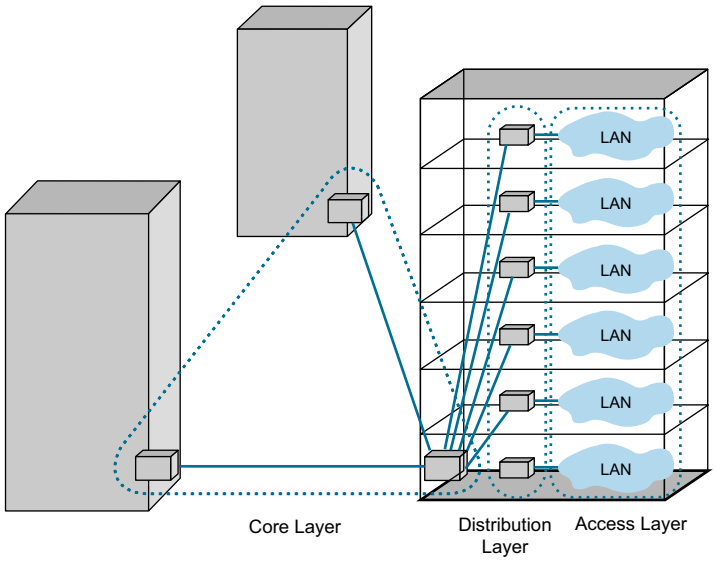


FIGURE 10.22

Typical VLAN system architecture and the VLAN backbone architecture layers: core layer, distribution layer, and access layer.

(3) VLAN standards

The IEEE 802.1Q specification establishes a standard method for tagging Ethernet frames with VLAN membership information. It defines the operation of VLAN bridges that permit the definition, operation and administration of virtual LAN topologies within a bridged LAN infrastructure. The IEEE 802.1Q standard is intended to address the problem of how to break large networks into smaller parts so broadcast and multicast traffic do not take more bandwidth than necessary. The standard also helps provide a higher level of security between segments of internal networks.

The key for the IEEE 802.1Q to perform the above functions is in its tags. Compliant switch ports can be configured to transmit tagged or untagged frames. A tag field containing VLAN (and/or IEEE 802.1p-specified priority) membership information can be inserted into an Ethernet frame. If a port has an 802.1Q-compliant device attached (such as another switch), these tagged frames can carry VLAN membership information between switches, thus letting a VLAN span multiple switches. However, it is important to ensure ports with non-802.1Q-compliant devices attached are configured to transmit untagged frames. Many of network interface cards (NICs) for PCs and printers are not 802.1Q-compliant. If they receive a tagged frame, they will not understand the VLAN tag and will drop the frame. Also, the maximum legal Ethernet frame size for tagged frames was increased in 802.1Q (and its companion, IEEE 802.3ac standard) from 1518 to 1522 bytes. This could cause network interface cards (NICs) and older switches to drop tagged frames as oversized.

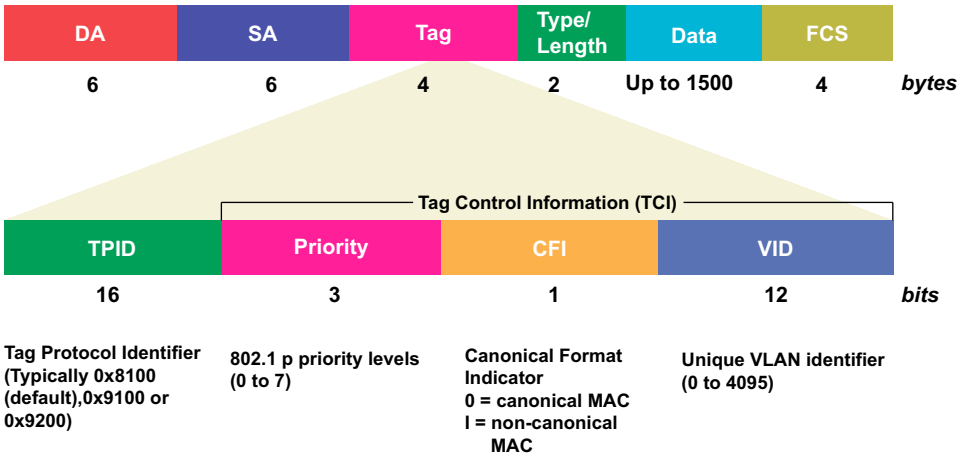


FIGURE 10.23

IEEE 802.1Q specification establishes a standard for tagging Ethernet frames with VLAN membership information.

Figure 10.23 shows the frame structure specified by the IEEE 802.1Q standard, which includes the following parameters:

(a) TPID defines the value of 8100 in hex. When a frame has the Ethernet type equal to 8100, this frame carries the tag IEEE 802.1Q and 802.1P.
(b) TCI means tag control information. This field includes user priority, canonical format indicator and VLAN ID.
(c) User priority has eight levels. IEEE 802.1P defines the operation for these three user priority bits.
(d) CFI means canonical format indicator, which is always set to zero for Ethernet switches. CFI is used for compatibility reasons between an Ethernet-type network and a token-ring-type network. If a frame received at an Ethernet port has CFI set to 1, then that frame should not be forwarded as it is to an untagged port.
(e) VID is VLAN ID, which is the identification of the VLAN, as used by the standard 802.1Q. It has 12 bits and allows the identification of 4096 VLANs. Of the 4096 possible VIDs, a VID of 0 is used to identify priority frames and the value 4095 (FFF) is reserved, so the maximum possible number of VLAN configurations is 4094.

10.4.4 Wireless local area network (WLAN)

Wireless networks are becoming more and more popular in industrial applications, including manufacturing and process automation, chemical and food production, and railway and highway transport systems, etc. since they offer a high degree of flexibility and lower installation and operation costs. Wireless networks have the same capabilities and comparable speeds to a wired network without the difficulties associated with laying wire, drilling into walls, or stringing Ethernet cables throughout an office, a building, or even an enterprise.

In industry, extremely reliable products, providing mechanisms for real-time support (guaranteed transmission times) and deterministic characteristics (predictable data traffic) are usually needed. Devices such as programmable controllers (PLCs) must transfer their data reliably even in critical situations. Such reliability is also achieved by optimum planning and installation and robust construction of the wireless link.

Whatever requirement the industries have, it is important that the products used are standardized and do not include proprietary procedures. An open standard provides a high degree of data security. Industrial wireless LAN uses only mechanisms that are precisely defined in the related IEEE (Institute of Electrical and Electronic Engineers) standards. For this reason, this textbook does not include detailed information on this topic and the reader is referred to specialist books and documents on IEEE 802.11 standards, including IEEE 802.11, 802.11a, 802.11b, and 802.11g.

(1) Wireless networking system components

Like a wired LAN, a wireless LAN is a grouping of computers, workstations and other programmable intelligent devices that share a common communication linkage. As is implied by the name, a wireless LAN allows users to connect to the LAN wirelessly via radio-wave transmission. The following are the most common components of an industrial wireless LAN.

(a) Access point

An access point is the point of attachment to a wired LAN. It increases the effective range of a wireless network and provides additional management and security features. Access points are very useful for larger networks, and they are particularly well-suited for adding wireless capability to an existing wired network (wireless networks of three or fewer nodes or users do not require an access point for an ad hoc networking mode). Some types of wireless access point can connect via an RJ-45 cable to a LAN and can support up to 20 wireless users at an effective range of up to 1500 feet in open spaces. It also enables additional security features such as MAC address authentication.

(b) Wireless PC card

A wireless PC card is used with integrated antennae to link a network node (for example a field device) to an industrial wireless LAN network. Some wireless PC cards allow for ad hoc networking of up to three nodes at an effective range of up to 1000 feet in open spaces.

(c) Wireless PCI adapter

A wireless access PCI adapter with integrated antennae allows desktop computer users to access the LAN. Some wireless PCI adapters allow for ad hoc networking of up to three desktop nodes at an effective range of up to 1000 feet in open spaces.

(d) Wireless router

A wireless router with built-in wireless access point can function in a wired LAN, a wireless only LAN or a mixed wired and wireless network. Most wireless routers have LAN ports which provide functions like the ports of a network switch. The routing functions are filtered using this port. If it is not used, many functions of the router will be bypassed. Furthermore, most wireless routers have wireless antennae which allow connections from other wireless devices such as network interface cards (NICs), wireless repeaters, wireless access points, and wireless bridges, etc.

(2) Wireless networking operational modes

The IEEE 802.11 specification defines two types of operational modes for wireless networks: ad hoc (peer-to-peer) mode and infrastructure mode.

In ad hoc mode, the wireless network consists of IEEE 802.11 network interface cards (NICs) only; thus a very simple architecture, as illustrated in the upper panel of Figure 10.24. In ad hoc mode, also known as independent basic service set (IBSS), or peer-to-peer mode, all of the nodes connected with a wireless NIC card can communicate with each other via radio-waves without an access point. The ad hoc mode is convenient for quickly setting up a wireless network, in locations where sufficient wired infrastructure does not exist.

In infrastructure mode, the wireless network is composed of both a wireless access point(s) and IEEE 802.11 network interface cards (NICs). The access point acts as a base station in an IEEE 802.11 network and all communications from all of the wireless clients go through the access point.

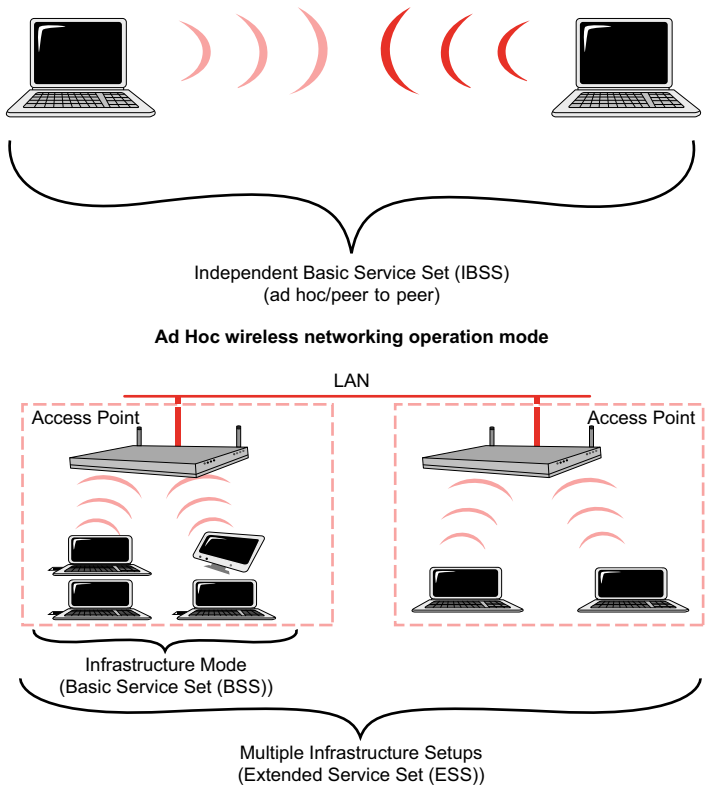


FIGURE 10.24

Two wireless networking operational modes: ad hoc mode and infrastructure mode.

(Courtesy of U.S. Robotics.)

The access point also provides for increased wireless range, future growth of the number of wireless users, and additional network security. In infrastructure mode, the data frames travelling from the LAN to a wireless client are converted by the access point into radio-wave signals and transmitted out into the environment. All wireless clients and devices within range can receive the frames, but only those clients with the appropriate destination address will receive and process the frames. A basic wireless infrastructure with a single access point is called a basic service set (BSS). When more than one access point is connected to a network to form a single sub-network, it is called an extended service set (ESS). These features are described in the lower panel of Figure 10.24.

(3) Wireless networking technical issues

As wireless networking becomes increasingly useful, several technologies have emerged, including narrowband, spread spectrum, frequency hopping spread spectrum, and direct sequence spread spectrum.

(a) Narrowband

Narrowband technology uses a specific radio-wave frequency (in the range of 50 cps to 64 kbps) for data transmission.

(b) Spread spectrum

Spread spectrum technology allows for greater bandwidth by continually altering the frequency of the transmitted radio-wave signal, thus spreading the transmission across multiple frequencies. Spread spectrum uses more bandwidth than narrowband, but the transmission is more secure, more reliable, and easier to detect.

(c) Frequency hopping spread spectrum (FHSS)

FHSS technology synchronizes the changing frequency of both the transmitter and receiver (using a narrowband carrier) to, in effect, produce a single transmission signal. This frequency hopping can occur as often as several times a second; it changes between frequencies, transmitting data for a certain time before changing back again. Like spread spectrum technology, FHSS technology requires additional bandwidth, but over multiple carrier frequencies.

(d) Direct sequence spread spectrum (DSSS)

DSSS technology breaks down the transmitted stream of data into small pieces across a frequency channel. A redundant bit pattern (known as a chipping code) is generated for each bit transmitted. Generally, the longer the chipping code, the more likely it is that the original transmitted data will be properly received. DSSS technology uses more bandwidth than FHSS, but is considered more reliable and resists interference. Because of the chipping code, data can still be recovered without retransmission of the signal, even in the case of damaged data bits.

Since wireless networking uses radio-wave transmission, radio-wave frequency techniques are particularly important to the wireless networks' performance. There are three technical issues are particularly relevant; frequency allocation, frequency roaming, and frequency interference.

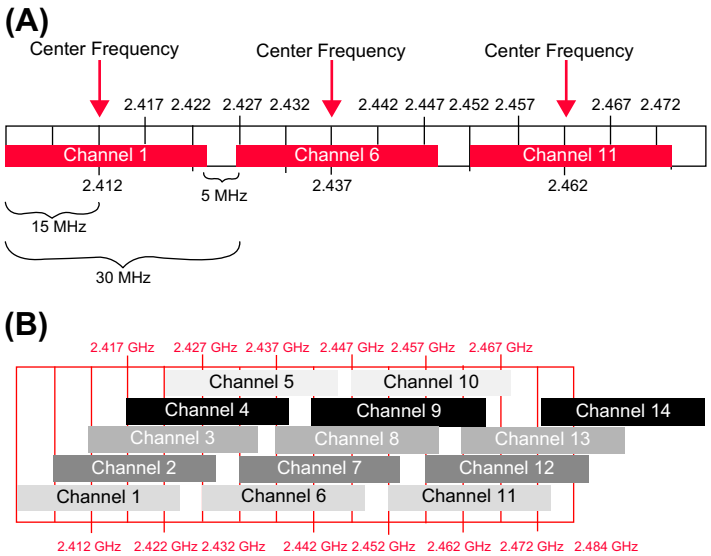


FIGURE 10.25

The frequency allocation techniques for wireless networking. (A) Bandwidth required for some of the channels specified in IEEE 802.11 standards; also demonstrates the 5 MHz between each channel. (B) An example of overlapping frequencies if based on the IEEE 802.11b bandwidth allocation.

(Courtesy of U.S. Robotics.)

(a) Frequency allocation for wireless networking

The IEEE 802.11b standard defines 14 frequency channels (Figure 10.25(A) gives some of these channels' bandwidths), governmental restrictions of frequency allocations in wireless networking apply in certain countries. In North America, the Federal Communications Commission and Industry Canada allow manufacturers and users to use channels 1 through 11; most of Europe can use channels 1 through 13, while in Japan, users have all 14 channels available. The actual channel frequency indicates the center frequency used by the transmitter and receiver for communication. An IEEE 802.11b radio signal consumes approximately 30 MHz of frequency spectrum, leaving a 5 MHz separation between center frequencies. This means that the signal extends out by 15 MHz from the center frequency spectrum. As a result, each channel signal overlaps several adjacent frequencies. This leaves the typical US user with three channels available for use by access points (channels 1, 6, and 11) that are within radio range of adjacent access points, which is explained in Figure 10.25(B).

(b) Frequency roaming for wireless networking

The IEEE 802.11 specification includes frequency roaming capabilities that allow a client node to roam among multiple access points on different channels. Thus, roaming client nodes with weak signals can associate themselves with other access points with stronger signals. Alternatively, by

setting up multiple access points to cover the same geographic area and by using different non-overlapping frequencies, client workstation networking loads can be better balanced. The NIC of a wireless LAN may automatically decide to re-associate itself with another access point within range if the load on its current access point becomes too high for optimal performance. Figure 10.26 shows how roaming among access points with non-overlapping frequencies allows for virtually unlimited coverage range in wireless networking.

(c) Frequency interference in wireless networking

The IEEE 802.11b standard uses the unlicensed radio-wave spectrum that is commonly shared by a variety of consumer devices such as baby monitors and cameras, 2.4 GHz cordless phones, microwave ovens, and Bluetooth-enabled devices. This can impact wireless LAN performance by generating radio-wave frequency interferences. DSSS technology is very effective at minimizing these types of interference. A quick scan around a user's area will indicate whether there are any potential frequency interference problems. Using products designed to work in the 900 MHz frequency range can help minimize any frequency interference and maximize the performance of any wireless LAN.

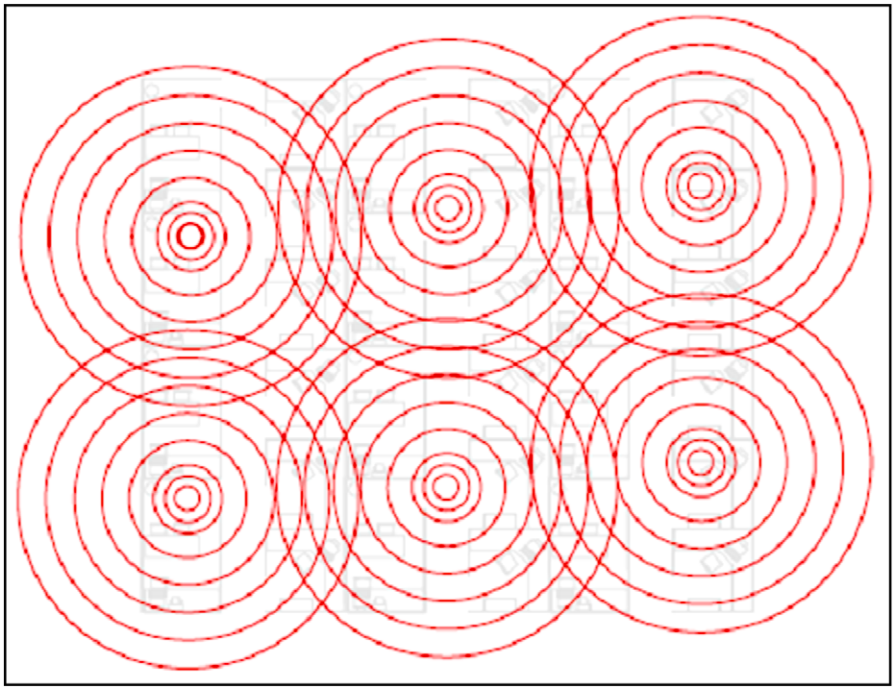


FIGURE 10.26

A possible roaming among access points with non-overlapping frequencies allows for virtually unlimited coverage range existing in wireless networking.

(4) Wireless networking industrial solutions

If wireless communication is used for production and manufacture, the reliability of the wireless channel becomes an important issue. In contrast to a consumer environment, machine downtimes involve high costs. Industrial wireless LAN provides the following solutions to enhance industrial wireless networking reliability.

Two methods are available to monitor the wireless channel.

(a) IP-Alive: IP-alive monitors the cyclic communication connections common in automated engineering environments. On selected IP connections, the monitoring function checks whether frames are actually exchanged at the times prescribed by the cycle time. If frames are missing, errors can be reported in various ways (error LED, log file, E-mail, SNMP trap). The user can configure the method of reporting to suit the situation.

(b) Link check: when link check is used, if there is no communication taking place, frames are sent to the node at selectable cyclic intervals to check its presence in the wireless network. This method somewhat reduces performance but provides a high degree of operating reliability.

In communication networks, a significant amount of traffic is generated by multicasts and broadcasts. If such messages dominate the bandwidth, there is a danger that productive data traffic will be restricted. This problem must be handled in the devices themselves. To achieve this, the "storm threshold" method is used to prevent overload and to restrict multicast and broadcast traffic to a minimum value.

Diversity antennae can be used to achieve a more reliable wireless link in a difficult environment in which reflections and multiple path reception interfere with radio-wave propagation. The use of antenna diversity is easy to recognize because two antennae are available for one wireless NIC card. The recipient can then evaluate the information from them both and select the better one dynamically during reception. When transmitting, the diversity function automatically selects the other antenna after a set number of failed attempts.

Problems

1. From those CAN systems you are familiar with, such as the CAN systems for cars or electric lifts, try to work out: (a) which CAN networks need a CAN repeater; (b) which CAN networks need a CAN bridge; (c) which CAN networks need a CAN gateway.
2. Based on the CANopen device model given in Figure 10.3, try to plot a data flow diagram for a CAN network using the CANopen protocol.
3. Suppose that in Figure 10.4, node 3 and node 4 are sending their own frames (base CAN message defined in Table 10.2) to the CAN bus simultaneously. Specify all the fields' values and the changes in their messages and explain the bus arbitration mechanism.
4. Do you agree with that if any distributed control network fully implements these four functions: data acquisition, data presentation, data communication and system control, this network can be a SCADA network?
5. Please explain why the master slave model can be combined with the peer to peer model by using two types of slot in the data communication of SCADA networks.
6. Please give the precise definition of firmware in industrial control networks.
7. Are there some substantial differences between the DNP3 and the IEC 60870 5 for SCADA networks in (a) communication models such as the polled communication model (master slave) and the contention communication model (peer to peer)? (b) Are there substantial differences in transmission methods such as unicast, multicast, and broadcast?

8. For the SCADA networks of electrical power grids over such countries as China, America, and Canada, which communication protocol, DNP3 or IEC 60870 5, would be better for network communication?
9. What are the differences between SCADA communication security and SCADA system reliability?
10. With reference to Figure 10.13, search the Internet for devices such as an injector and a picker used in the "adapting devices for PoE using power injector and power picker" technique. Then, based on IEEE 802.3af, give the steps of this PoE process.
11. With reference to Figures 10.12 and 10.15, say whether or not the spanning tree protocol for Ethernet network redundancy should be implemented by layer 2 (data link) or layer 3 (network) of the OSI model.
12. Do you think an industrial enterprise network should be the same as a distributed industrial control system? Can we say that an industrial enterprise network is actually a distributed control system with additional functions for remote accesses and information management?
13. In industrial control networks, why are the CSMA/CD protocols used for CAN, SCADA and LAN networks?
14. In Figure 10.21, please briefly descibe how LAN emulation operates in the transmission of a frame.
15. Do you agree that the ATM based VLAN uses transport layer oriented protocols according to the ISO/OSI Reference Model?
16. Please give details of why the wireless networks of three or fewer nodes do not require an access point for the ad hoc networking mode?

Further Reading

Konrad Etschberger. Controller Area Network: Basics, Protocols, Chips and Applications. IXXAT Automation GmbH. Germany. 2001.

Keith Pazul, Microchip Technology Inc. Controller area network (CAN) basics. http://www.chipdocs.com/manufacturers/MCHIP.html. Accessed: June 2008.

Tomas Waggershauser, IXXAT Automation GmbH. Advantages of CAN topology components in building automation systems. http://www.ixxat.com/index en.html. Accessed: June 2008.

H. Boterenbrood, NIKHEF. CANopen High level protocol for CAN bus. March 2000.

NI (National Instruments). Controller area network (CAN) overview. http://zone.ni.com/dzhp/app/main. Accessed: June 2008.

Steve Corrigan, Texas Instruments Inc. (http://www.ti.com). Controller Area Network Physical Layer Require ments: Application Report SLLA270. January 2008.

Steve Corrigan, Texas Instruments Inc. (http://www.ti.com). Introduction to the Controller Area Network (CAN): Application Report SLOA 101. August 2002.

Cobus Strauss. Practical Electrical Network Automation and Communication Systems. Elsevier (Newnes), London. 2002.

Gordon Clarke, Deon Reynder, Edwin Wright. Practical Modern SCADA Protocol: DNP3, 60870.5 and Related Systems. Elsevier (Newnes), Sydney. 2002.

David Bailey. Practical Radio Engineering and Telemetry for Industry. Elsevier (Newnes), Sydney. 2002.

Wikipedia (http://en.wikipedia.org). SCADA. http://en.wikipedia.org/wiki/SCADA#searchInput. Accessed: June 2008.

John Tritak, Riptech, Inc. (http://www.riptech.com). Understanding SCADA System Security Vulnerabilities. January 2001.

Cisco (http://www.cisco.com). White paper (C11 450264 00 01/08): Industrial Ethernet: A Control Engineer's Guide. Accessed: July 2008.

IXXAT (http://www.ixxat.de). Product White Paper: Industrial Ethernet Module. Accessed: July 2008.

Rockwell Automation (http://www.rockwellautomation.com). 2001. EtherNet/IP: Industrial Protocol White Paper. Accessed: July 2008.

Altair (http://www.altair.org). POE Power over Ethernet. http://www.altair.org/labnotes POE.html. Accessed: July 2008.

Contemporary Controls Ltd (http://www.ccontrols.co.uk). Info Sheet: Ethernet Redundancy. July 2008.

John P. Slone (Ed.). Local Area Network Handbook (6th edition). Auerbach, London. 1999.

InetDaemon (http://www.inetdaemon.com). Tutorials on local area networks. http://www.inetdaemon.com/tutorials/networking/lan/index.shtml. Accessed: July 2008.

Chaim Ziegler. Local area networks, 2004. http://www.sci.brooklyn.cuny.edu/~ziegler/CIS49.2/LANS.pdf. Accessed: July 2008.

Javvin Technologies, Inc.(http://www.javvin.com). VLAN and protocols; WAN and protocols. http://www.javvin.com/protocolLAN.html. Accessed: July 2008.

David Passmore, John Freeman (DeciSys, Inc.). The virtual LAN technology report. http://www.3com.com/other/pdfs/solutions/en US/20037401.pdf. Accessed: July 2008.

U.S. Robotics (http://www.usr.com). Wireless LAN Networking White Paper. July 2008.

CHAPTER

Networking devices 11

Networking devices can also be called networking equipment or networking components. They serve a variety of roles to split, switch, boost, or direct digital data or signals along a network. Products include hubs, switches, routers, bridges, gateways, repeaters, and firewalls.

Hubs provide a central location for attaching wires to network nodes, which can be computers and programmable controllers, etc. Passive hubs do not amplify signals, whereas active hubs serve to extend the wire connection to more network nodes.

A network switch is a small hardware device that connects network nodes together, or allows a large number of nodes to share a limited number of ports.

Routers are protocol-dependent network devices that connect two or more logical subnetworks or network segments.

Bridges, similar to hubs and repeaters, interconnect local or remote networks and operate at the physical and link layers of the open systems interconnection reference model.

A network gateway is an internetworking system capable of joining together two networks that use different base protocols. A network gateway can be implemented completely in software, completely in hardware, or as a combination of both. Depending on the types of protocols they support, network gateways can operate at any level of the open systems interconnection reference model.

Repeaters are IEEE 802.3 network cable segments which use regeneration and retiming to ensure that a signal is transmitted clearly through all subnetworks.

Firewalls are a system or group of systems that, for purposes of security, enforce an access control policy between an enterprise network and the Internet.

The functionality of network devices can be specified in accordance with the OSI reference model. Hubs and repeaters work at the first, physical layer; bridges and switches are thought of as network devices working at the second, data-link layer; routers work at the third, network layer; gateways are the most complex devices with respect to the functionality and work at the fourth layer (transport layer) or the uppermost layers of the OSI reference model. Figure 11.1 is a diagram to illustrate the possible correspondence between these networking devices and the OSI model layers. Hubs and repeaters are network devices without intelligent networking functions, but switches, bridges, routers and gateways have different degrees of network intelligence.

Network devices also use protocols to specify the software attributes of data communications, including the structure of packets. Choices include: asynchronous transfer mode (ATM), controller area network bus (CANbus), control network (ControlNet), DeviceNet, fiber channel, fiber distributed data interface (FDDI), frame relay, integrated services digital network (ISDN), synchronous optical network (SONET), token bus and token ring, etc. Ethernet is a popular protocol that uses a bus or star typology. Types include: 10Base-T or twisted pair Ethernet, 10Base-2 or 10/100 Ethernet, 100Base-T or Fast Ethernet, and Gigabit Ethernet. Network equipment is often defined by protocol or port type. Serial port or asynchronous serial interfaces are system-to-system communication interfaces

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10011-7

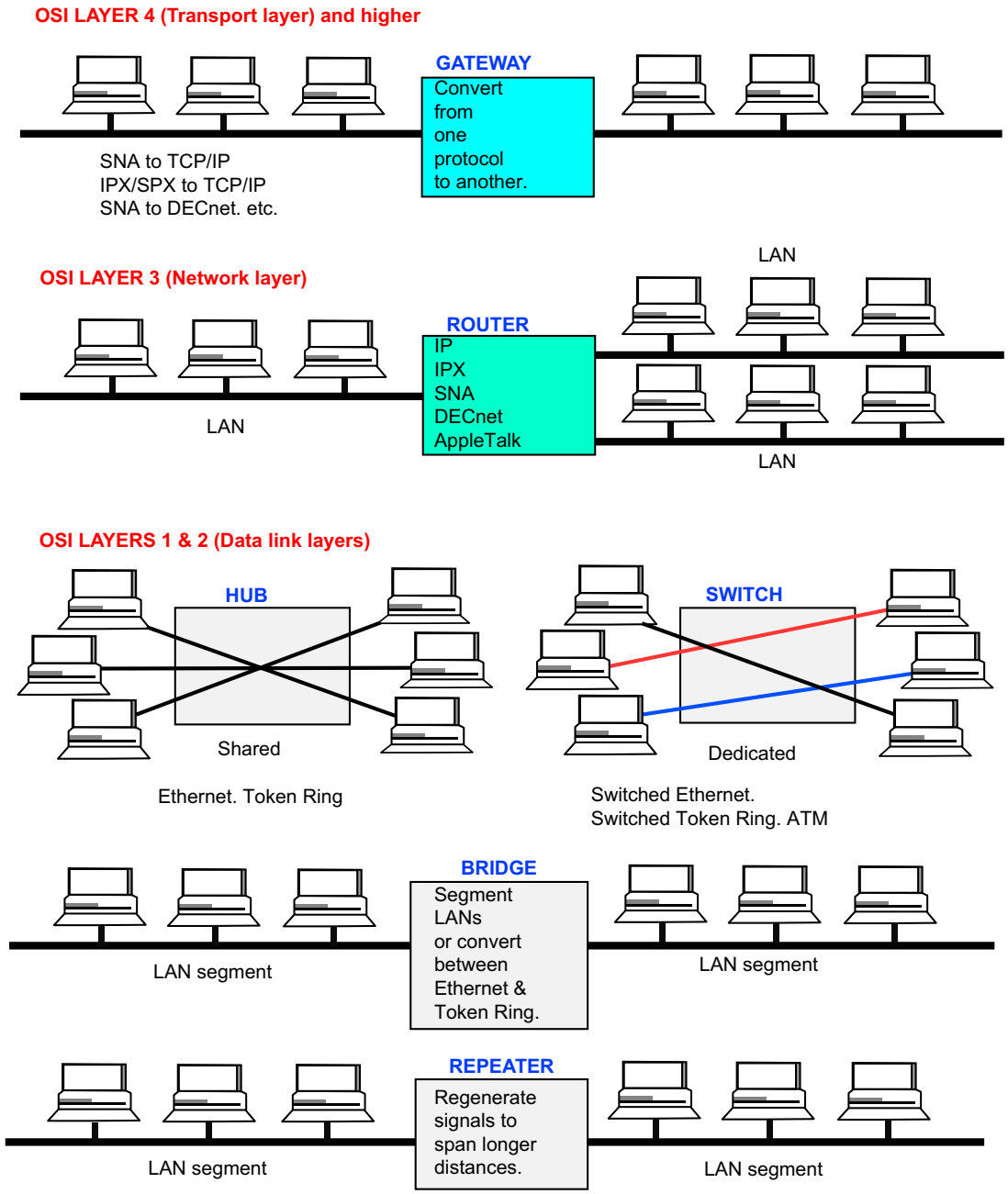


FIGURE 11.1

The possible correspondences between some networking devices (gateways, routers, hubs, switches, bridges, and repeaters) and the OSI model layers.

such as RS232, RS422, and RS485. Selecting network equipment requires an analysis of parameters such as area network type, form factor, memory, performance, and features. There are three types of enterprise network: local area network (LAN), metro area network (MAN), and wide area network (WAN). Form factor choices include chip, board and module. Memory specifications include random access memory (RAM), synchronous dynamic random access memories (SDRAMs), and Flash memory. Data rate, operating temperature, and number of concurrent connections are additional specifications to consider. In terms of features, some products are stackable, rack-mounted, hardened, or suitable for virtual LAN (VLAN). Others have an alarm or indicator, provide IP addressing, and have an integral firewall for security. Network equipment should meet the Restriction of Hazardous Substances (RoHS) directive from the European Union (EU). Full-duplex products can transmit data in both directions.

11.1 HUBS AND SWITCHES

11.1.1 Overview

A hub (sometimes referred to as a concentrator) is to a network device which receives a data packet from one network node and retransmits to all other connected nodes. In its simplest form, a hub works by duplicating the data packets received at its entry port and making them available to all other ports, therefore allowing data sharing between all devices connected to the hub. It also centralizes network traffic coming from multiple hosts, and propagates the signal. Therefore, a hub needs enough ports to link machines to one another, usually 4, 8, 16 or 32 (Figure 11.2(A) shows some hubs). As with a repeater, a hub operates on layer 1 of the OSI reference model, the physical layer, which is why it is sometimes called a multiple-port repeater.

A switch (sometimes named a switching hub) refers to a network device which filters and forwards data packets across a network. A switch is normally a multiple-port device (it can have 48 of more ports; Figure 11.2(B) shows some switches), meaning that it is an active element working on layer 2 of the OSI model. Unlike a standard hub which simply replicates what it receives, a switching hub keeps a record of the medium access control (MAC) addresses of the devices attached to it. When the switch receives a data packet, it forwards it directly to the recipient device by looking up the MAC address. The switch analyses the frames coming in on its entry ports and filters the data in order to focus solely on the right ports; as a result, it can act as both a port when filtering and as a hub when handling connections. A network switch can utilize the full throughput potential of a network connection for each device, making it preferable to a standard hub.

Some discussion about how hubs and switches perform their functions in networks is presented here.

(1) A network hub or repeater is a fairly unsophisticated broadcast device. They do not manage any of the traffic that comes through them, and any packet entering any port is broadcast out on every other port except for the entry port. If two or more nodes try to send packets at the same time, a collision is said to occur, and the network nodes have to go though a routine to resolve the conflict. The process is prescribed by the Ethernet Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol. Each Ethernet adapter has both a receiver and a transmitter. If the adapters did not have to listen with their receivers for collisions they would be able to send data at the same time they are receiving it (full duplex). Because they have to operate at half duplex (data flow one way at a time) and a hub retransmits data from one node to all of the nodes, the maximum bandwidth is shared by all of the nodes connected to the hub.

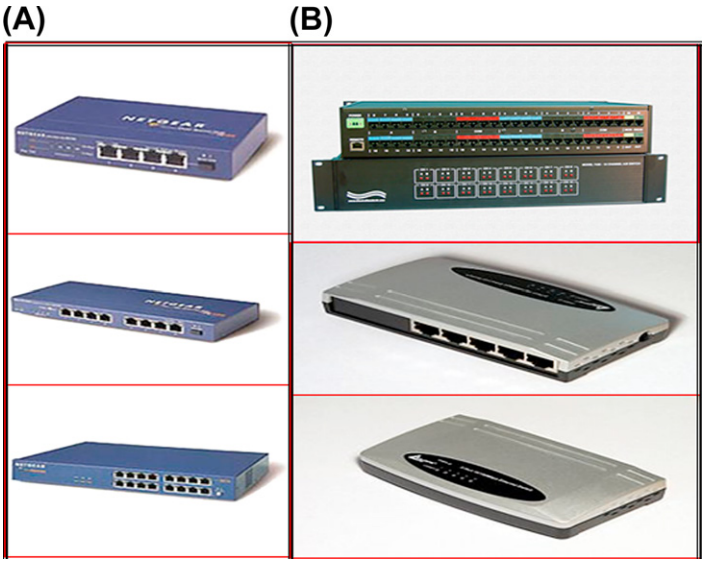


FIGURE 11.2

Some hubs and switches used in networks: (A) from the top down are a 4-port hub, an 8-port hub, and a 16-port hub; (B) from the top down are a wireless controllable network switch, a typical network switch for small office, and an Ethernet switch.

A hub network behaves like a shared medium, that is, only one device can successfully transmit at a time and each host remains responsible for collision detection and retransmission. Some hubs have special stack ports, allowing them to be combined in a way that allows more hubs than simple chaining through Ethernet cables, but even so a large Fast Ethernet network is likely to require switches to avoid the chaining limits of hubs. It is possible to connect several hubs together in order to centralize a larger number of machines; this is sometimes called a daisy chain. To do this, the hubs are connected using crossover cable, a kind of cable which links the input and output ports on one end to those on the other. Hubs generally have a special port called an uplink for connecting two hubs together using a patch cable. There are also hubs which can cross or uncross their ports automatically depending on whether they are connected to a host or a hub.

(2) A network switch typically includes a set of input ports for receiving packets arriving on the buses, a set of output ports for forwarding packets outward on the buses, and a switch fabric such as a cross-point switch for routing packets from each input switch port to the output switch ports that are to forward them. Network switch input and output ports often include buffer memories for storing packets until they can be forwarded thorough the switch fabric or outward on a network bus. An output port's buffer allows it to receive data faster than it can forward it, at least until the buffer fills up. When the buffer is full, incoming data are lost. A network switch port often uses one or more SDRAMs to implement its buffer memory since they are inexpensive. Some input switch ports include protocol processors for converting each incoming packet to a sequence of cells of uniform size. The input port stores the cells in its buffer memory until it can forward them through the switch fabric to one of the

output ports. Each output switch port in turn stores cells received via the switch fabric in its buffer memory and later forwards them to another protocol processor which reassembles them into a packet to be forwarded outward on a network bus.

The traffic manager in a network switch forwards packets of the same flow in the order that it receives them, but may forward high-priority packets preferentially. An output switch port may include a traffic manager for storing the cells received via the switch fabric in its buffer memory and for forwarding them later to another protocol processor. The output port's protocol processor reassembles each cell into a packet and forwards the packet outward on a network bus. A network switch output port may be able to forward a packet outward to another network switch or to a network node on a selected channel of any of several different network buses, and the traffic manager of a network switch output port decodes the packet's a flow identification number (FIN) to determine which output bus or bus channel is to convey the packet away from the port. The output port's traffic manager may also decode a packet's FIN to determine a packet's minimum and maximum forwarding rate and priority. The network switch also includes an address translation system which relates a network destination address to an output port that can forward the packet to that network address. When an input port receives an incoming packet it stores it, reads its network destination address, consults the address translation system to determine which output port is to forward the packet, and then sends a routing request to the switch's arbitration system.

Hubs and switches are used to divide up networks into a number of subnetworks. For example, if a plant floor is dynamically exchanging large amounts of data across the network, its traffic will slow down the network for other users. To solve this problem, two switches can be used, with a floor's computers being connected to form one network while the remaining computers are connected to form another. The two switches can then be connected to the router that sits between the internal network and the internet. The floor's traffic is only seen by the computers on that network, but if they need to connect to a computer on the other network, data are sent through the router in the middle.

Modern networking equipment combines the multiple connectivity of the hub with the selective routing of data packets from different protocol networks with the help of bridges (see section 11.3). Modern switches also have plug-and-play capability. This means that the switches are capable of learning the unique addresses of devices attached to them, even if those devices are plugged into a hub which in turn is then attached to the switch, without any programming. If a computer or an industrial controller is plugged directly into a switch, that switch would only allow traffic addressed to that device to be sent to it. By controlling the flow of information between ports, switches achieve major advantages over current shared environments:

(a) When all devices are directly connected into a switch port, the opportunity for collision between ports is eliminated. This ensures that packets arrive with much greater certainty than in a shared environment.

(b) Each port has more bandwidth available to it. In a shared environment, any port in the system could consume the entire bandwidth at any given time. This means that during a traffic peak, the network availability of any other node is greatly reduced. In a completely port-switched environment, however, the only traffic flowing down the wire between any node and the switch is either traffic destined for, or created by, that particular node.

In conclusion, switches and hubs provide industrial users with much of the functionality that could only be provided by wiring distinct, proprietary control networks in the past. The elimination of

collisions by connecting every node to a switched port, coupled with the hub ability to keep control and office traffic from interacting unwontedly, while still using one physical network, allows industrial users to enjoy the open architecture and massive bandwidth and speed of Ethernet without compromising the integrity of their control traffic.

11.1.2 Network hubs

There are many types of network hubs with various features and specifications, which fall into four main types: passive hubs, active hubs, intelligent hubs, and USB hubs.

(a) Passive hubs. A passive hub simply receives data on input ports and broadcasts it on the output ports without even rectifying it. As the name suggests, passive hubs work simply as an interface between the networking topologies. They do not rectify or enhance the data or signals they pass on in the network, thus they do not enhance network performance. It is very hard to get any help from passive hubs while troubleshooting if there is any fault in the hardware or the network.

(b) Active hubs. An active hub participates in data communication within the enterprise or local area networks. Active hubs come with various features, such as the ability to receive the data or signal from the input port and store it before forwarding it, which allows the hub to monitor the data it is forwarding; some have a feature that helps in preferentially transmitting data of high priority; some can synchronize data communication by retransmitting the data packets that are not properly received at the receiving computer or by adjusting re-transmission of the data packets to compensate timing; some active hubs come with a feature that rectifies the data or signal before forwarding them. Active hubs also help in troubleshooting at a certain level by identifying bottlenecks within the network.

(c) Intelligent hubs. In addition to having all the features of passive and active hubs, an intelligent hub also provides some features to help in managing data communication within the network. An intelligent hub can decide which data packet goes in which output line, which helps in controlling and minimizing data traffic in the network. An intelligent hub recognizes the slower devices automatically and helps them to transmit the data at their own speed. This feature also improves the performance of the network manifolds. An intelligent hub also adapts to changes in the network very easily and it also supports different technologies without the need to change anything in the configuration. As an active hub helps in finding out where problems are, but an intelligent hub locates the problem in the network, diagnoses it and tries to rectify it without letting the problem hamper the performance of the network. Intelligent hubs provide features that help in determining the exact cause and location of the fault, which saves a lot of time and energy.

(d) USB hubs. A universal serial bus (USB) hub is used in a network for connecting a set of computer peripheral apparatus to a host computer. All USB devices attach to the USB via a port on specialized USB devices known as hubs. A USB hub is an intelligent wiring connector coupled to a networking device to allow attachment of peripheral devices. USB hubs are also wiring concentrators that enable multiple attachment characteristics converting a single attachment point into multiple attachment points. The USB hub is connected to the computer via a single upstream connector. The upstream port of a hub connects the hub to the host. The USB hub also includes a plurality of downstream ports for connecting the peripheral devices to the hub. Each of the other downstream ports of a hub allows connection to another hub or function. The USB hub uses a standardized connector at the downstream ports to provide universal connectivity between peripheral devices and the computer. USB hubs can

detect attaching and detaching at each downstream port and enable the distribution of power to the downstream devices. Each downstream port can be individually enabled and configured as either full or low speed. The hub also isolates low-speed ports from full-speed signalling ports. The hub relays data from the computer to all enabled devices coupled to the data hub, and relays data from the enabled devices to the computer without any data storage or significant delay. A USB hub typically includes a hub controller, a hub repeater, and a transaction translator. The hub repeater provides a USB protocol-controlled switch between the upstream port and downstream ports as well as support for reset and suspend/resume signaling. The host controller facilitates communication to and from the host. The USB hub carries out its internal functions and distributes power supplied to other hubs and functions connected further down.

Protocol, port, and features are the most important specifications to consider for network hubs. Protocol is the fundamental mechanism for network communications; they specify the software attributes of data communications including the structure of a packet and the information contained in it. They may also prescribe all or some of the operational characteristics of the hardware on which they will run. UDP and TCP/IP are the most popular, but ATM, CANbus, ControlNet, DeviceNet, 10 Base-2 Ethernet, 10/100 Ethernet, 10Base-T Ethernet, Fast Ethernet, Gigabit, Ethernet, Fiber Channel, FDDI, Fieldbus, Frame Relay, Interbus, ISDN, Profibus, SONET, Token Ring, and xDSL are also available.

The number of ports and port type need to be specified. The number of ports determines the total number of ports available on the networking device. Port types can be IEEE 1394, ISDN, or USB. Common features for network hubs include stackability, rack mounting, and LED indicators. Network hubs that are stackable conserve space. Rack mount hubs are designed to be mounted into a rack. LED indicators are used to indicate the status of the hub.

11.1.3 Network switches

Network switch is a broad and imprecise marketing term for a computer networking device that connects network segments. The term does not generally encompass unintelligent or passive network devices such as hubs and repeaters. They may operate at one or more OSI layer, including physical, data link, network, and transport (end-to-end). A network device that operates simultaneously at more than one of these layers is called a multilayer switch, although use of the term is diminishing. However, most network switches route packets between ports at the OSI layer 2, which means that (in Ethernet) the network switches decide where incoming packets are transferred to, based on the 48-bit address of the network interface cards. Upon receipt of a packet, the switch forwards the packet to its destination port.

(a) Unmanaged switches. These switches have no configuration interface or options. They are typically found in small office or home environments.

(b) Managed switches. This type of switch allows access to one or more interfaces for the purpose of configuration or management of features such as spanning tree protocol, port speed, virtual LANs, etc. High-end or "enterprise" switches may provide a serial console and command-line access via telnet and Secure Shell; they may provide management via SNMP; and may also provide a website interface. Managed switches are found in enterprise networks where their management usually requires understanding of layer 2 networks such as Ethernet.

(c) Intelligent switches. These are managed switches with a limited set of features. Likewise, "website-managed" switches are switches which fall between unmanaged and managed. Intelligent

switches can provide a website interface and allow configuration of basic settings, such as virtual LANs, port-speed and duplex.

Assessment of the performance of a digital network switch is often based on metrics such as the number of ports, the total bandwidth (or number of digital bits per second) that can be switched without blocking or slowing the data traffic, and the memory volumes and its speeds for read and write data buffers. The number of ports specifies the total number of ports available on the networking device. Data rate is the maximum data transfer speed. Memory is the total memory of the network device. Common port configurations for network switches include IEEE 1394, ISDN, and USB. Features for network switches include stackability, rack mounting, LED indicators, and full duplex. Switches that are stackable conserve space. Full duplex signifies the ability of a device or line to transmit data simultaneously in both directions. As with hubs, Ethernet implementations of network switches support either 10/100 Mbit/second or 10/100/1000 Mbit/second ports Ethernet standards. Large switches may have 10 Gbit/second ports. Switches differ from hubs in that they can have ports of different speed.

The network switch plays an integral part in most Ethernets or LANs. Mid-to-large-sized LANs contain a number of linked managed switches. Small office, home office (SOHO) applications typically use a single switch. However, fast switches are becoming more and more popular in Ethernets or LANs because of the reasons below.

(a) Controlled Ethernet traffic. In a hub, a frame is passed along or broadcast to every one of its ports, even though it is only destined for one port. The hub has no way of distinguishing which port a frame should be sent to, so passing it to every port ensures that it will reach its intended destination. This places a lot of traffic on the network and can lead to poor network response times. A switch, on the other hand, keeps a record of the MAC addresses of all the devices connected to it, so it can identify which system is sitting on which port. Hence, when a frame is received, it knows exactly which port to send it to, which significantly increases network response times.

(b) Bandwidth utilization. A 10/100Mbps hub must share its bandwidth with all ports, so when only one node is broadcasting, it will have access to all of the available bandwidth. If, however, multiple nodes are broadcasting, then that bandwidth will need to be divided between all of those systems, which will degrade performance. A 10/100Mbps switch will allocate a full 10/100Mbps to each of its ports. So regardless of the number of PCs transmitting, users will always have access to the maximum amount of bandwidth. It is for these reasons that a switch is considered to be a much better choice than a hub.

(c) Manageability. Hubs are considered "dumb" devices. They are strictly plug-and-play and are utilized to piece together an Ethernet network for simple IP communications. Hubs cannot be managed as switches can. Switches have the option of being unmanaged or managed, layer 2 or 3. Today's switches can also act as routers and are called layer 3 switches with most of the functions you find in routers. Each individual port can be configured for a specialized network, such as one for VoIP.

11.2 NETWORK ROUTERS

11.2.1 Overview

A network router is a computer networking device that forwards electronic digital data or signals within a network or between separate networks. It will be connected to at least two networks or at least

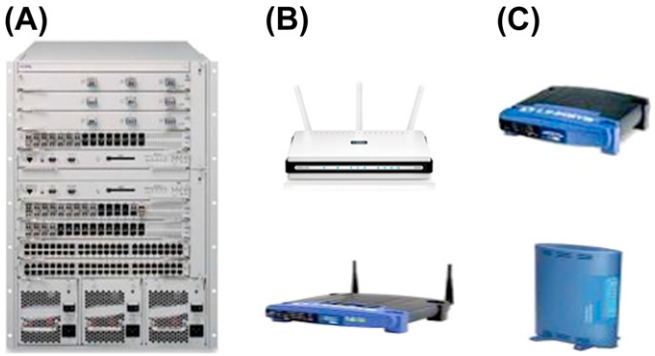


FIGURE 11.3

Some routers used in networks: (A) an 1800px-ERS-8600 router made by Nortel Corporation for industrial applications; (B) two wireless broadband routers for home and small office use; (C) two wired broadband routers for home and small office use.

two subnetworks that can be LANs (local area network), WANs (wide area network), corporate or enterprise networks, Internets or ISP (Internet service provider) networks, and home or small office networks. As mentioned before, routers are assumed to work with the layer 3, the network layer, of the OSI model. The term layer 3 switch is thus used interchangeably with network router, but switch is really a marketing term without a rigorous technical definition. Figure 11.3 shows some network routers.

A router takes information from multiple sources and routes it to multiple destinations within networks or subnetworks, typically being used to send data packets to the destination's data terminal equipment. A network is a collection of interconnected electronic devices which allow users to access resources and data. Networks can include the nodes themselves, a connecting medium (wired, wireless and/or a combination of wired and wireless), and network switching systems such as routers, hubs and/or switches. The nodes typically communicate by exchanging discrete frames or packets of data according to predefined protocols. In a packet-based network, such as the Internet, the computing devices communicate by dividing the data into small blocks called packets which are individually routed across the network from source to destination. The destination device extracts the data from the packets and assembles it back into its original form. Dividing the data into packets enables the source device to resend only those individual packets that may be lost during transmission.

Traditional routers are designed to join multiple area networks (LANs and WANs). On the Internet or on a large corporate network, for example, routers serve as intermediate destinations for network traffic. Routers for home networks (often called broadband routers) are designed specifically to join the home (LAN) to the Internet (WAN) for the purpose of Internet connection sharing. These routers receive TCP/IP packets, look inside each packet to identify the source and target IP addresses, and then forward them to their final destination. However, in the larger company or enterprise, routers are also used to separate LANs into subnetworks in order to balance traffic within workgroups, and to filter traffic for security purposes and policy management. Here, routers serve as an internet backbone that connects all internal networks, usually via Ethernet. Within the global Internet, routers do all the packet switching between the backbones and are typically connected via ATM (asynchronous transfer

mode) or SONET (synchronous optical network). Furthermore, routers often contain a built-in firewall for security, which serves all users in the network without requiring that the personal firewall in each computer be turned on and configured.

Figure 11.4 shows the sites where routers can be located in the computer network to perform their functionality i.e. in between LANs; in the core; for Internet or for remote access.

A network router is a more sophisticated network device than either a switch or a hub. The functions of a router, hub and a switch are all quite different from one another, even if at times they are all integrated into a single device. Where a hub or switch is concerned with transmitting data packets,

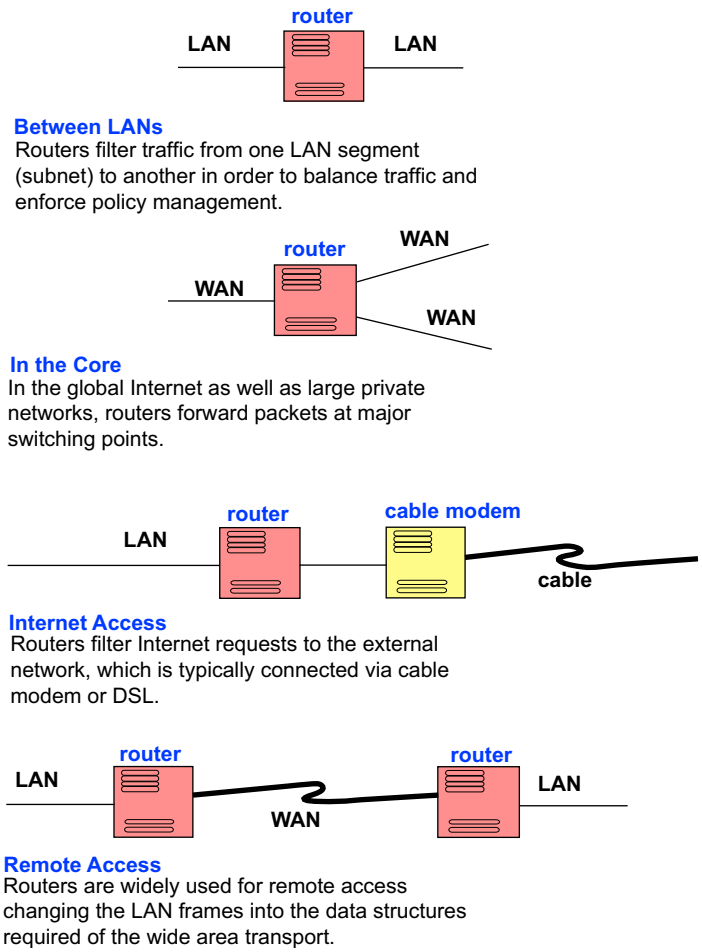


FIGURE 11.4

The possible sites where routers can locate in networks to perform their functionality: they can be in between LANs; they can be in the core; they can be for Internet access; they can be for remote access.

(Courtesy of Computer Desktop Encyclopaedias by the Computer Language Company, Inc., 2005.)

the job of a router is to route packets to other networks until that packet ultimately reaches its destination, given by address of where it is going held within the packet.

In contrast, neither hubs nor switches are capable of joining multiple networks or sharing an Internet connection. A home network with only hubs and switches must designate one computer as the gateway to the Internet, and that device must possess two network adapters for sharing, one for the home LAN and one for the Internet WAN. With a router, all home computers connect to the router equally, and it performs the equivalent gateway functions. Routers, like bridges, provide bandwidth control (also termed data flow control) by keeping data out of subnetworks where they do not belong. However, routers, once set up, can communicate with other routers and learn the way to parts of a network that have been added after it was initially configured.

Routers differ slightly from network to network, so are now available in many types, though all fundamentally play the same role in a network. What defines a router is not its shape, color, size or manufacturer, but how it routes data packets between network computers and devices. Table 11.1 lists all the types of routers used today.

11.2.2 Router operations, specifications and configurations

Routers operate in two different phases; the control phase and the forwarding phase and require supporting hardware and software. Router hardware physically connects separate computer networks, and is responsible for routing the network information from source to destination. Router software is used to determine the point-to-point switching in the network information path, and is used to determine each path taken on the network. When used in a network, routers need to be configured, to verify that their hardware and software can adapt to the network.

(1) Router operation phases

Routers implement their operations in terms of two work phases: the control phase and the forwarding phase.

(a) Control phase

The control phase constructs the routing table from knowledge of the up and down status of its local interfaces, from hard-coded static routes, and from exchanging routing protocol information with other routers. It is not compulsory for a router to use routing protocols to function, if for example it was configured solely with static routes. Several different information sources may provide information on a route to a given destination, but the router must select the "best" route to install into the routing table. In some cases, there may be multiple routes of equal quality, and the router may install all of them and load-share across them.

In routing, the control phase is concerned with drawing the network map, or the information in a (possibly augmented) routing table that defines what to do with incoming packets. Control phase logic can also define certain packets to be discarded, as well as preferential treatment of certain packets for which a high quality of service is required by such mechanisms as differentiated services.

A major function of the control phase is to decide which routes go into the main routing table. Here the term main refers to the table that holds the active unicast routes. If the router also does multicast routing, there may be an additional routing table for these routes.

Table 11.1 Types of Networking Routers

<table>
<tr><td colspan="2" rowspan="2">Software-based Routers
Software-based routers are based on a computer server environment to perform network routing services, in conjunction with other functions.</td><td colspan="4">Embedded routers
Embedded routers may be a stand-alone computer system, dedicated to its IP router functions. Alternatively, it is possible to embed router functions within a host operating system that supports connections to two or more networks.</td></tr>
<tr><td colspan="4">Server routers
Server routers provide a solution that meets the needs of terminal server networking requirements today.</td></tr>
<tr><td rowspan="6">Hardware-based Routers
Hardware-based routers are self-contained device units. The hardware routers have their on-board processor that provides packet handling and routing.</td><td rowspan="6">Wired; Wireless routers; Or hybrid routers</td><td rowspan="3">Routers for Internet connectivities and internal usages</td><td>Edge routers</td><td>Provider edge router;
Subscriber edge router;
Inter-provider border router.</td><td rowspan="6">ADSL and DSL routers
ADSL modem or DSL modem is a device used to connect a single computer or router to a DSL phone line, in order to use an ADSL service
Broadband routers
A broadband router combines the features of a traditional network switch, a firewall, and a server. Broadband routers are designed for convenience in setting up home networks.</td></tr>
<tr><td>Core routers</td><td>The routers that reside within the middle or backbone of the LAN network rather than at its periphery.</td></tr>
<tr><td rowspan="3">Routers for enterprise networks</td><td>Node routers</td><td>Node routers are the main routers linking the networks</td></tr>
<tr><td>External routers</td><td>External routers establish a connection between autonomous networks.</td></tr>
<tr><td>Internal routers</td><td>Internal routers allow routing of information inside an autonomous network.</td></tr>
<tr><td rowspan="3">Routers for virtual private networks (small or home offices)</td><td>Node routers</td><td>The same as above</td></tr>
<tr><td>External routers</td><td>The same as above</td></tr>
<tr><td>Internal routers</td><td>The same as above</td></tr>
</table>

(b) Forwarding phase

This performs the pure Internet Protocol (IP) forwarding function. In routing, the forwarding phase decides what to do with packets arriving on an inbound interface. Most commonly, it refers to a table to look up the destination address of the incoming packet and retrieves the path from the receiving element, through the internal forwarding fabric of the router, and to the proper outgoing interface(s).

As mentioned above, the routing table also might specify that the data packet is discarded. In some cases, the router will return an ICMP (Internet Control Message Protocol) "destination unreachable" or other appropriate code. Some security policies, however, dictate that the router should be programmed to drop the packet silently, in order to keep a potential attacker from becoming aware of a target that is being protected. Depending on the specific router implementation, the routing table in which the destination address is looked up could be the routing table, or a separate forwarding information base that is populated (i.e., loaded) by the control phase, but used by the forwarding phase to look up packets and decide how to handle them. Before or after examining the destination, other tables may be consulted to make decisions to drop the packet based on other characteristics, such as the source address, the IP protocol identifier field, or TCP or UDP port number.

In general, the passage from the input interface directly to an output interface, through the fabric with minimum modification at the output interface, is called the fast path of the router. If the packet needs significant processing, such as segmentation or encryption, it may go onto a slower path, which is sometimes called the services phase of the router.

(2) Router hardware components

Routers are nothing more than a special type of personal computer (PC). Routers and computers both have some of the same components, such as a microprocessor, motherboard, RAM, interface boards, and an operating system. The main difference between a router and a standard PC is that the former performs special tasks to control or "route" traffic between two or more networks. There are seven major internal components of a router: CPU, RAM, NVRAM, Flash, ROM, console, and interfaces. The view and block diagram of a router system board are given in Figure 11.5.

(a) CPU

The CPU performs the same functions as in a normal PC. It executes commands given by the IOS (input/output system) by using other hardware components. High-end routers may contain multiple processors or extra slots to add more CPUs.

(b) RAM

The main roles of the RAM are to hold the ARP (address resolution protocol) cache, to store routing tables, to hold fast-switching cache, to perform packet buffering, and to hold queues. It also provides temporary memory for the configuration file of the router while it is powered on. However, the RAM loses content when the router is restarted or powered off. Please note that this component is upgradeable.

(c) NVRAM

Nonvolatile RAM is used to store the start-up configuration files. This type of RAM does not lose its content when the router is restarted or powered off.

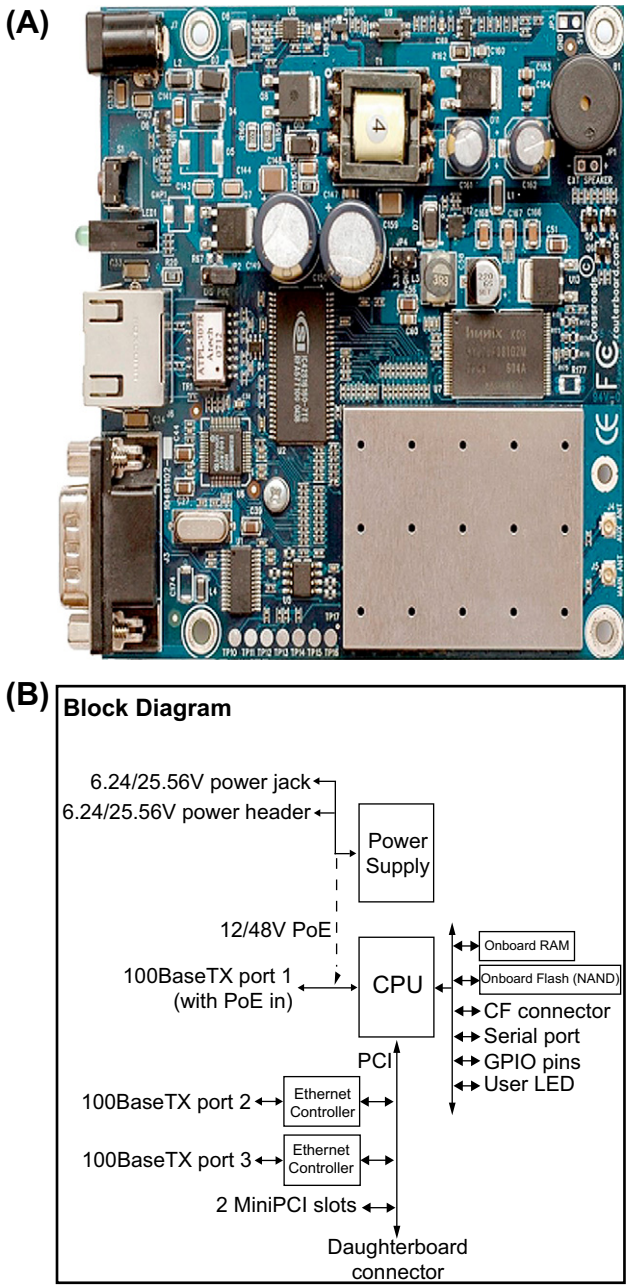


FIGURE 11.5

The system board of a network router: (A) system board view; (B) system board block diagram.

(Courtesy of Cisco Systems, Inc.)

(d) Flash

Flash memory saves data flows if the operating system configuration is disrupted. It holds the IOS image file, as well as backups. This flash memory is classified as an EEPROM (electronically erasable programmable read only memory), and is upgradeable in most Cisco routers.

(e) ROM

The ROM performs the same operations as the BIOS (basic input/output system). It holds information about the system's hardware components and runs POST (power on self test) when the router first starts up. This component can be upgraded by unplugging the chip and installing a new one.

(f) Console

The console consists of the physical plugs and jacks on the router. It provides access for configuration.

(g) Interfaces

The interfaces provide connectivity to LAN, WAN, and console. They can be jacks soldered onto the motherboard, transceiver modules, or card modules. Routers, especially the higher-end models, can be configured in many different ways. They can use a combination of transceivers, card modules and onboard interfaces.

(3) Router software components

A router's software package must include a real-time operating system (RTOS) to provide the basic command functions for the routing device as well as various subsystem components. In addition to the RTOS, routers generally organize their software into a management plane, input/output plane, control plane, and forwarding plane.

(a) Management plane

The management plane oversees the router in processing its routing functions. Its responsibilities include configuring the forwarding paths, scheduling other planes' operations, monitoring the packet routing status, and handling possible traffic congestion and data errors.

(b) Input/output plane

The input/output plane works as a device driver to manage input/output interfaces at each driver and slot level, and as a handler for configuration, status reporting and statistics. It is typically embedded into the RTOS of the router, to ensure that it requires modest CPU and memory resources.

(c) Control plane

The control plane is the largest and most challenging software component in a router, since it is responsible for the implement of routing protocols, data-link layer protocols, and gateway to off-box services. This software component must, therefore scale in terms of physical and logical (subscriber) interface count, routing protocol peers, and route and prefix counts, and the off-box services transactions per second. This component is also required to have a very rich feature set with high feature velocity, and some real-time capabilities.

(d) Forwarding plane

The forwarding plane manages the hardware for the forwarding function including building tables, trees, classification tables, etc. as required to process packets and apply features. It also gathers statistics and state from forwarding plane hardware. It can require significant CPU and memory resources because scale, performance and real-time response are important requirements.

(4) Router performance specifications

Specification parameters include number of ports, data rate, and memory. Number of ports specifies the total number of ports available on the networking equipment. Data rate is the maximum data transfer speed. Memory is the total memory capacity of the network equipment. Common port choices for network routers include IEEE 1394, RJ-45, serial, ISDN, and USB. Features for network routers include stackability, rack mounting, LED indicators, integrated firewall, IP addressing, and VPN (virtual private network). Routers that are stackable conserve space. Rack mount network routers are designed to be mounted into a rack. LED indicators are used to indicate the status of the router. An integrated firewall is used for extra security. IP addressing is used for smarter routing. A VPN is a connection that has the appearance and many of the advantages of a dedicated link but occurs over a shared network. Using a technique called tunneling, data packets are transmitted across a public routed network in a private "tunnel" that simulates a point-to-point connection and allows network protocols to traverse incompatible infrastructures.

(5) Basic router configuration

A typical router includes ports for channeling communication throughout the network, a primary port facility having a single processor, and a router card for controlling the router ports. Packets are received at an inbound port, and ultimately forwarded from an outbound port. A basic router configuration includes a chassis which contains basic components such as power supply, and slots where interface cards and network modules are inserted. The line cards are inserted into card slots and modules are inserted into module slots to handle packet ingress and egress and other networking functions. A routing table is initially downloaded to each line card along with configuration files. One of these files, the dispatch table, is specifically constructed for suitability for use within a local routing switch processor in the line card. Line cards provide one or more interfaces through which traffic flows. Depending on the number of slots and interfaces, a router can be configured to work with a variety of networking protocols.

Routers play a major role in any network and their basic configuration is vital to ensure they run efficiently. The basic configuration of network routers includes two necessary contents; IP address configuration and routing protocol configuration.

(a) The IP address configuration will offer the router a correct IP address so that the router has its identification in the network. Because it is located between two or more networks or subnetworks, a router is an interface with the respective connected networks. The IP address configuration is thus to configure the IP address of an interface by means of the interface type slot and port, or interface type port to enter the interface configuration mode.

(b) The routing protocol configuration sets up the Routing Information Protocol (RIP), a vector distance type protocol, in the router (particularly ensuring that the correct version of the RIP is used). Each router communicates with the other routers within a specific distance (the number of hops which separates them). So, when a router receives a messages it increments this distance by

1 and sends it to directly accessible routers. In this way, the routers can then optimize the route of a message by storing the next router address in the routing table in such a way that the number of hops to reach a network is kept to a minimum. However, this protocol only takes into account the distance between two machines in terms of hops and does not consider the state of the connection so as to select the best possible bandwidth (data rate).

To allow for configuration, the router manufacturers provide IOS software that accesses several different command modes, each providing a seperate group of related commands. For security purposes two levels of access, user and privileged are provided. The unprivileged user mode is called user execution mode, and the privileged mode is called privileged execution mode and requires a password. The commands available in user execution mode are a subset of those available in privileged execution mode. There is one mode, configuration mode, which is crucial to set the configuration. It has a set of submodes that are used for modifying interface settings, routing protocol settings, line settings, and so forth. This mode should be used with caution because all changes take effect immediately.

Any changes to the router configuration must be saved into the router's system memory, as otherwise they will be lost if there is a system reload or power outage. There are two types of configuration files: the running (current operating) configuration and the start-up configuration.

11.2.3 Protocols and algorithms for network routing

Routing is a technique for selecting the optimal path from a number available in order to transmit an IP packet to a final destination. A typical router includes various interfaces that send and receive packets and packets originating from various source locations are received via a plurality of communication interfaces. Each packet generally includes a header containing a destination address, which allows the router to route it, or send it directly to its destination. Each packet contains routing information, such as a source address and a destination address, which are associated with the respective communication interface of the router. The router receives a given packet through the first interface, processes the packet to determine how to best forward the packet to its destination, and then transmits the packet through a selected second interface. It uses the destination address to decide the next-hop information of the packet. High-speed routers make these decisions at several million packets per second. Each search finds the longest prefix match of the destination address among all stored prefixes in the router.

Routers are necessary when a network is subdivided into subnetworks and so each subnetwork is also a separate broadcast domain. If a packet is destined for a host on a different network or subnetwork, it must be forwarded through the local router. The router uses this way out onto another network or subnetwork.

A computer network typically includes a collection of routers interconnected to each other. Based on the destination address contained in the header portion of the received packet, the router searches its forwarding table, and determines an interface number for transmitting the packet, and forwards the packet to the determined interface. Routers typically process information packets, in the order received, so that packets exit in a router in the same order as they entered it. A routing protocol is used to exchange information with other routers in order to maintain a consistent view of the network. For packets to be forwarded properly, each router must have a consistent FIB (forwarding information table) with other routers on the network. Many popular types of routers operate under the control of

packet processing software, which manipulates the individual packets to be forwarded. To reduce network disruption due to router failure, some networks employ virtual routers, which are also known as virtual router groups. Virtual routers typically comprise two or more routers which share the same IP address.

Routers use protocols to communicate, and to report changes so that routers can share information and act accordingly. Thus, a network can adapt to changes in a dynamic way. A protocol is an elemental formula that the router uses in order to appropriately determine the path that data must be sent from one router to another. There are basically two kinds of communication protocols used by routers: routable (or routed) protocols, and routing protocols.

(1) Routable (routed) and non-routable (non-routed) protocols

Data sent along a path chosen from several available ones means that the data is routed, and the protocols that support such communications are known as routable protocols (or routed protocols). Such a protocol contains both a network address and a device address, and allows packets to be forwarded from one network to another. Examples of routable protocols are TCP/IP, Internet Protocol (IP), Telnet, Remote Procedure Call (RPC), SNMP, SMTP, and AppleTalk.

Non-routable protocols (or non-routed protocols) contain only a device address and not a network address. They do not incorporate an addressing scheme, and so cannot support communications outside a local network. Usually, non-routable protocols work only on a similar type of network; for instance, NetBEUI (Network BIOS Enhanced User Interface) is non-routable protocol and it will work only on Microsoft Networks.

Because routable protocols can be used to link several networks together and create new wide-area environments, they are becoming increasingly important. Routable protocols are larger protocols than the non-routable ones because they involve many types of error-checking, which increase the size of data-packets and makes them slower.

Since they serve limited capabilities, non-routable protocols are extremely fast. If you plan a small isolated network, non-routable protocols will need to be considered. However, if your concern is also the Internet or any other network, non-routable protocols will not serve you and routable protocols are the best choice.

(2) Routing tables

A routing table, or routing information base (RIB), is an electronic file or database-type object that is stored in a router or a networked computer, holding the routes (and in some cases, metrics associated with those routes) to particular network destinations. This information contains the topology of the network close to it. Construction of these tables is the primary task of routing protocols and static routes. They are generally not used directly for packet forwarding in modern router architectures; instead, they are used to generate the information for a smaller forwarding table which contains only the routes which are chosen by the routing algorithm as preferred routes for packet forwarding, often in a compressed or pre-compiled format that is optimized for hardware storage and lookup.

An example routing table is given in Figure 11.6. In part A, the first row of the table states that network 10.0.10.0 is on interface “Eth0” and the “C” in the “Learned” column means the information was learned through a direct connection; in other words, the network is local to the router. The last row states that network 10.0.21.0 is on interface “S0” and the “R” in the “Learned” column means the information was

(A)

Learned	Network Address	Hop	Interface
C	10.0.10.0	0	Eth0
C	10.0.11.0	0	Eth1
C	200.200.4.0	0	S0
R	10.0.20.0	1	S0
R	10.0.21.0	1	S0

(B)

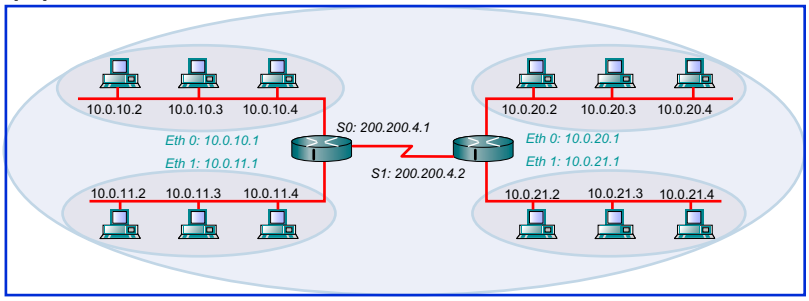


FIGURE 11.6

The routing tables used for network routing by network routers: (A) an example of a routing table; (B) an illustration of the correspondent network topology.

learned through a routing protocol; in this case, the network is not local to the router. Figure 11.6(B) illustrates the network topology correspondent to the routing table given in Figure 11.6(A).

(3) Routing protocols

The routing protocol calculates the best path for forwarding data and ensures that routing information is sent regularly between routers, so that network information can be shared. This allows routers to build up information about network topologies and keep information about routes. The routing protocol also specifies how routers report changes and share information with the other routers in the network. It allows the network to dynamically adjust to changing conditions, which would be very difficult with static data.

Confusion often arises between the terms routing protocols and routable protocols. While routing protocols are used by the router to make the decision on which paths to send data packets, routable protocols are responsible for the actual transfer of data packets between network devices. Specifically, a routable protocol is any network protocol that provides enough information in its network layer address to allow a packet to be forwarded from one host to another, on the basic of an addressing scheme, without knowing the entire path from source to destination. Routable protocols define the format and use of the fields within a packet. Packets generally are conveyed from end system to end system.

The routing protocol on one system can be different from that used on another. A routing protocol used within an autonomous system is referred to as an internal gateway protocol (IGP); examples of which are RIP, IGRP, EIGRP, IS-IS and OSPF.

For routing between each autonomous system, an exterior gateway protocol (EGP), is required. The most common EGP in current use is Border Gateway Protocol (BGP). The Internet Assigned Numbers Authority (IANA), the body that regulates the assignment of IP addresses, allocates a unique 16-bit number to each autonomous system, known as the autonomous system number. BGP uses this information to route traffic between autonomous systems.

A number of routing protocols have been developed over the years. The more common of these are described below.

(a) Routing Information Protocol (RIP)

This is one of the most frequently used routing protocols on internal networks. RIP is a distance-vector routing protocol that uses a hop count as its routing metric. If there are multiple paths to a destination, RIP chooses the path with the lowest hop count which is not necessarily the fastest, as Figure 11.7(A) illustrates. RIP comes in different versions; RIP-1 and RIP-2. The later version was developed to overcome some limitations of the first. More information is contained inside an RIP-2 routing packet than an RIP-1 packet and a subnet mask field is included. Thus, RIP-2 can support classless routing and hence a network divided into subnets with different subnet masks.

(b) Interior Gateway Routing Protocol (IGRP)

This is a proprietary routing protocol developed for overcoming the RIP's use of hop count as the routing metric and the hop count limit of 15 that made it unsuitable for use on larger networks. The maximum hop count of IGRP is 255, and like RIP, IGRP is a distance-vector routing protocol but it uses a more complex routing metric to determine path costs. Factors such as delay, bandwidth, hop count and reliability can be used in determining the fastest path, although delay and bandwidth are used by default. Figure 11.7(B) illustrates how the more complex metric of bandwidth and delay enables a superior path to be selected, compared to selection based solely on hop count.

(c) Extended Interior Gateway Routing Protocol (EIGRP)

This is also a distance-vector routing protocol, but is much more sophisticated than IGRP or RIP. To calculate the path costs, EIGRP uses a 32-bit metric which combines an assessment of the link bandwidth and delay. Routes are calculated more efficiently using an algorithm called the diffusing update algorithm. EIGRP has a higher rate of route convergence; in other words, routers exchange information and agree on routes more quickly. It will also query neighboring routers for information only when needed, so reducing the amount of traffic generated between routers. Unlike its predecessor, EIGRP is able to deal with classless routing, allowing the use of VLSM (variable length subnet masking).

(d) Open Shortest Path First (OSPF)

This is a protocol designed for large networks where protocols such as RIP may be unsuitable. It is a link-state routing protocol with an advanced metric algorithm; basing path selection on cost rather than hop count and without the limit of 15 hops in a route, hence allowing a network to grow beyond what RIP could support. With OSPF, a large network can be separated into smaller areas, confining

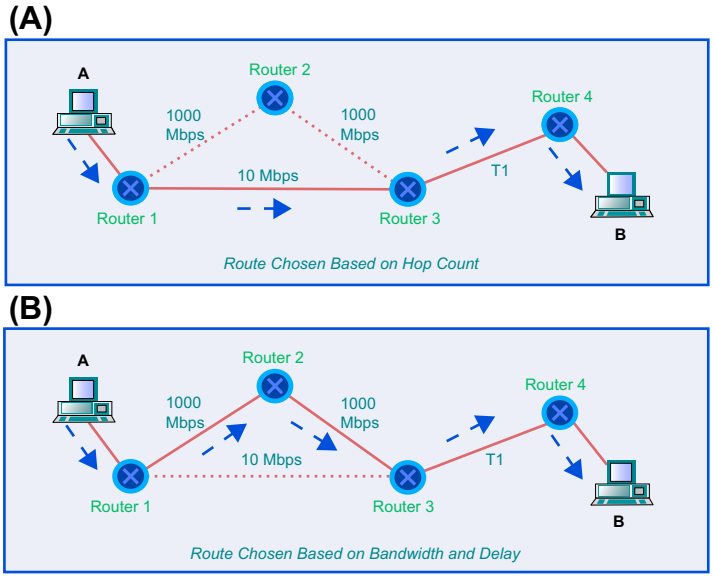


FIGURE 11.7

Two route-choosing algorithms used by routing protocols: (A) route chosen based on hop count; (B) route chosen based on bandwidth and delay.

many of the routing processes to each individual area, but allowing routers to communicate between areas. This cuts down on network traffic since a router does not have to recalculate its routing table unless a route in its own area changes. The separation of a network into areas is called hierarchical routing. In general, OSPF provides faster route convergence and generates less traffic between routers than RIP or IGRP.

(e) Intermediate System to Intermediate System (IS-IS)

This is a protocol developed as part of the OSI protocol stack, and is used to manage routing within Connectionless Network Protocol (CLNP) networks. Integrated IS-IS is a derivation of the original IS-IS which was developed to support the IP protocol. As a link-state routing protocol, IS-IS has the same basic features as OSPF, including link-state advertisement, where link-state information is sent out across the network, allowing routers to maintain a current network topology. Variable-length subnet masking (VLSM) is also supported by IS-IS.

(f) Border Gateway Protocol (BGP)

This is an example of an exterior gateway protocol (EGP), allowing routing information to be exchanged between autonomous systems. Autonomous systems are independent networks controlled by different organizations, each running an interior gateway protocol of their choice. For routing between each of these systems, another routing protocol, an exterior gateway protocol, is required.

(4) Routing algorithms

There are two terms, static routing and dynamic routing, which should be introduced here.

(a) Static routing

This is simply the process of manually entering routes into the routing table to be loaded when the routing device starts up. Since they do not change after they are configured, these routes are called static routes. This is the simplest form of routing, but it is a manual process and does not work well when information has to be changed frequently, or needs to be configured on a large number of routing devices (routers). Static routing also does not handle outages or down connections well, because any route that is configured manually must also be reconfigured manually to fix any lost connectivity.

(b) Dynamic routing

These protocols are software applications that dynamically discover network destinations and how to reach to them. A router will "learn" routes to all directly connected networks first. It will then learn routes from other routers that run the same routing protocol. It will then sort through its list of routes and select one or more "best" routes for each network destination it knows or has learned. Dynamic protocols will then distribute this information to other routers running the same routing protocol, thereby extending the information on what networks exist and can be reached. This gives dynamic routing protocols the ability to adapt to logical network topology changes, equipment failures or network outages on the fly.

Every routing protocol uses an algorithm to calculate the best path to a destination, and each routing algorithm uses metrics to calculate path costs. Metrics are simply values, such as hop count, bandwidth, delay, reliability and load. These ones used depend on the routing protocol, for example, RIP uses the hop count metric only, but IGRP uses bandwidth and delay by default. The metrics most commonly used by routing protocols are shown in Figure 11.8.

Each routing protocol will probably be categorized as either a distance-vector or link-state, which refers to the complexity of metrics used, and the regularity with which routers send routing table updates to their neighbors.

(a) The distance-vector routing approach. This uses the distance to a link in a network as its routing metric. Distance can be hops or a combination of parameters calculated to represent a distance value; the further the distance, the higher the overall metric cost. Examples of distance-vector routing

Hop Count:	The number of routers a packet has to travel through to reach its destination
Bandwidth	The capacity of a link, e.g. 10Mbps
Delay:	How long it takes to move a packet across the link; this is dependent on factors such as bandwidth, congestion and, physical distance
Reliability:	The error rate of the link
Load:	How much the link is in active use
Cost:	An arbitrary value that can be assigned by a network administrator

FIGURE 11.8

The metrics commonly used by routing protocols.

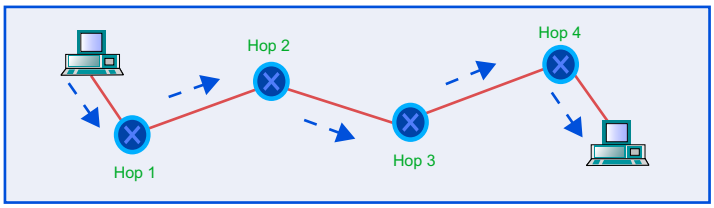


FIGURE 11.9

An illustration of the distance-vector routing protocol.

protocols and the metrics each one uses are: RIP v1 (metric: hops used); RIP v2 (metric: hops used); IGRP (metric: bandwidth and delay used). Whatever metrics are used to calculate distance, the path with the lowest cost is chosen as the best option. If hop count is used as the metric, then a path cost is the number of hops (routers) to the destination. All hops are seen as equal, regardless of their speed. Routes with the lowest hop count are preferred. Figure 11.9 gives an illustration of the distance-vector routing approach.

(b) The link-state routing approach. This also favors paths with the lowest cost within a network. Link-state routers have link information passed to them from neighbors; they then pass that information on, to other neighbors. Eventually, all the routers have information about all the links on the network and build a kind of topological map of the network. Link-state uses the shortest-path-first algorithm to calculate the best path to a destination. OSPF is an example of link-state routing protocol. However, whereas distance-vector periodically broadcasts its routing table to its neighbors, link-state only broadcasts every 30 minutes or so, or when the state of a link changes. If a link changes from up to down or vice versa, a notification called a link-state advertisement is flooded throughout the network. All the routers note the change, and recompute their routes accordingly. A router therefore passes information to the whole network, telling them about its neighboring links only. Link-state routing generates less network traffic, responds more quickly to changes in network topology and offers faster route convergence than distance-vector routing.

(c) Variable length subnet masking (VSLM). This is the extension of the standard IP class subnetwork masks to include subnetworks. A routing protocol categorized as classical does not send subnetwork mask information with route updates. This means that only the standard IP class network addresses are recognized and understood. A network containing subnetworks would encounter problems if such a protocol was used. A classless routing protocol sends subnet information with any routing information, which means that router can recognize subnetworks within a network.

11.3 BRIDGES, GATEWAYS AND REPEATERS

11.3.1 Network bridges

A network bridge is an electronic hardware device for carrying out bidirectional data transmission across a plurality of networks connected via ports. The network bridge is particularly suited to connecting two networks that have the same communication protocol, so allowing the bridge to segment a network. Secondly, a network bridge commonly has two connections between two distinct networks.

Thirdly, the bridge works at the layer 2, data-link layer, in the OSI reference model. Formally both in LAN and Ethernet networks, the term "bridge" means a device that behaves according to the IEEE 802.1D standard, although it is commonly referred to as a network switch in marketing language.

(1) Network bridging algorithms

Bridging is a forwarding technique used in packet-switched computer networks. Unlike routing, the bridging makes no assumptions about where in a network a particular address is located, but depends on broadcasting to locate unknown devices. In routing, once a device has been located, its location is recorded in a routing table to preclude the need for further broadcasting. Bridging depends on broadcasting, and is thus only used in local area networks.

Two different bridging technologies, transparent bridging and source routing, are in widespread use. Transparent bridging predominates in Ethernet networks, whereas source routing is used in token ring networks. Thus, bridging allows two networks to connect seamlessly on the data-link layer in the OSI model. In this respect, network bridges are like network switches.

(a) Transparent bridging

Also known as adaptive bridging, this refers to a form of bridging that is "transparent" to the end systems. The end systems operate as if the bridge does not exist, which leaves the bridge to segment broadcasts between networks, and only allow specific addresses to pass to the other network.

This bridging algorithm has been standardized as IEEE 802.1D. The bridging functions are confined to network bridges which interconnect network segments. The active parts of the network must form a tree, which can be achieved either by physically building the network as a tree, or by using bridges that use the spanning tree protocol. This is briefly discussed below.

The algorithm uses a forwarding database to send frames across network segments. This is initially empty, and entries in the database are built once the bridge receives frames. If an address entry is not found in the forwarding database, the frame is rebroadcast to all ports of the bridge, to all segments except for the source address. By broadcasting these frames, the destination network will respond and a route will be created. Along with recording the network segment to which a particular frame is to be sent, bridges may also record a bandwidth metric to avoid looping when multiple paths are available. Note that both source and destination addresses are used in this algorithm. Source addresses are recorded in entries in the table, while destination addresses are looked up in the broadcasting table and matched to the proper destination segment.

As an example, consider two host computers, A and B, and a network bridge C. The bridge has two ports or interfaces, C1 and C2. The host computer A is connected to C1; and the host computer B is connected to C2. The physical connection is from A to C and then to B, since the bridge C has two ports. The host computer A sends a frame to the bridge C, which records the source MAC address in its broadcasting table. The bridge C now has an address for the host computer A in its table, so it forwards it to the host B by broadcasting it to the MAC address (say FF:FF:FF:FF:FF:FF) or every address possible. It assumes that host B, having received a packet from A, now transmits a packet in response. This time, the bridge has A's address in the table, so it records B's address and sends it to A's unique MAC address specifically. Two-way communication is now possible between A and B without any further broadcasting. Note, however, that only the bridge along the direct path between A and B possesses table entries for B. If a third host computer, D, on the same side as A sends a frame to B, the bridge simply records the address source, and broadcasts it to B's segment.

(b) Source route bridging

This is based on Section 9 of the IEEE 802.2 standard, which does not need the structure of the spanning tree protocol. The operation of the network bridges is simpler, and much of the bridging functions are performed by the end systems, particularly the sources, giving rise to its name.

A field in the token ring header, the routing information field (RIF), is used to support source-route bridging. Upon sending a packet, a host computer attaches a RIF to the packet indicating the series of bridges and network segments to be used for delivering the packet to its destination. The bridges merely follow the list given in the RIF; if a given bridge is next in the list, it forwards the packet, and otherwise it ignores it. When a host wishes to send a packet to a destination for the first time, it needs to determine an appropriate RIF. A special type of broadcast packet is used, which instructs the network bridges to append their bridge number and network segment number to each packet as it is forwarded. Loops are avoided by requiring each bridge to ignore packets which already contain its bridge number in the RIF's field. At the destination, these broadcast packets are modified to be standard unicast packets and returned to the source along the reverse path listed in the RIF. Thus, for each route discovery packet broadcast, the source receives back a set of packets, one for each possible path through the network to the destination. It is then up to the source to choose one of these paths (probably the shortest one) for further communications with the destination.

As mentioned above, in the transparent bridging algorithm, a network bridge uses the IEEE spanning tree protocol to establish a loop-free forwarding topology. When there are multiple paths in a bridged network, loops can form, and the simple forwarding rules of a bridge can cause forwarding storms: a condition in which the same frame is relayed endlessly from one network segment, across the bridge, to another segment. The spanning tree protocol is an algorithm that provides an automated mechanism to selectively disable bridge forwarding on individual ports as is necessary, in order to ensure that the network topology is loop-free. No configuring of the network bridge is needed to use it.

The main idea of the spanning tree protocol is for the bridges to select the ports over which they will forward frames. Selection proceeds as follows: each bridge has a unique number as its identifier in a given LAN. The algorithm first elects the bridge with the "smallest identifier" as "the root bridge" of the tree in the given LAN. The root bridge always forwards frames out over all its ports. Next, each bridge computes "the shortest path" to the root and notes which of its ports is on this path. This port is also selected as the bridge's "preferred path" to the root. Finally, all the bridges connected to the given LAN elect a single "designated bridge" that will be responsible for forwarding frames towards the root. Each designated bridge in the LAN is the one that is closest to the root; if two are equally close to the root, then their identifiers are used to break ties, with the smallest being chosen. Of course, each bridge is connected to more than one LAN, so it participates in the election of a designated bridge for each LAN it is connected to. In effect, this means that each bridge decides whether it is the designated bridge relative to each of its ports. The bridge forwards frames over those ports in the designated bridge.

(2) Network bridge applications

Network bridges connect one network to another so that terminals on both of them can communicate as if they were part of a single network. They provide the connectivity required for communications between dissimilar segments of a given LAN. Normally, network bridges send packets between two networks of the same type; receiving packets from a segment connected to one port and forwarding

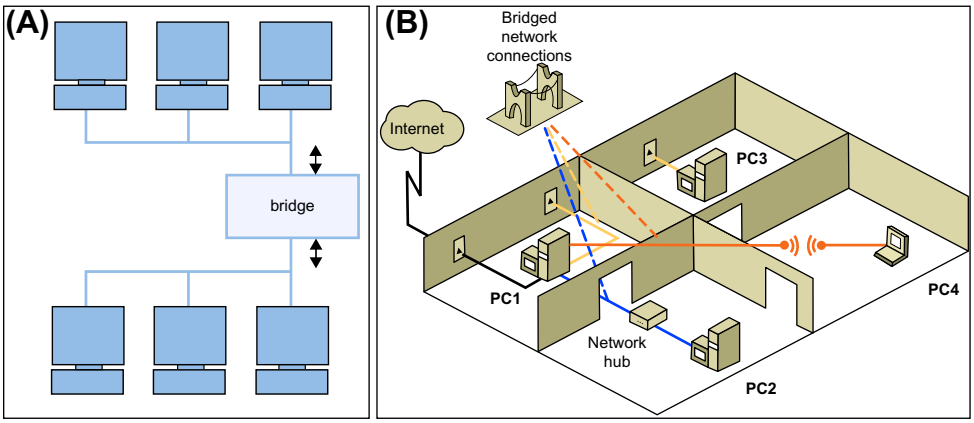


FIGURE 11.10

Some applications of network bridges: (A) a general topology of the networks using a bridge to connect two segments of the same type; (B) an example of a small office network connected by a network bridge.

them to another. If a LAN bridge is used to extend the network range, it also relieves the problem of congestion that multiple devices can cause on a single Ethernet segment. Figure 11.10(A) shows this function of network bridges.

The bridge is used to segment a network, holding back the frames intended for the LAN while transmitting those meant for other networks. This reduces traffic (and especially collisions) on all networks, and increases the level of privacy, because information intended for one network cannot be received on another.

Figure 11.10(B) supposes a small office network with four computers (PC1, PC2, PC3, and PC4) and one Ethernet hub. The four computers are running Windows XP; Windows Server 2003, Standard Edition; or Windows Server 2003, Enterprise Edition; and have the following hardware installed:

(a) PC1 has an adapter connecting it to the Internet, an Ethernet network adapter, an HPNA (Home Phone Network Alliance) network adapter, and a wireless adapter;
(b) PC2 has an Ethernet network adapter;
(c) PC3 has an HPNA network adapter;
(d) PC4 has a wireless network adapter.

The Ethernet adapters on PC1 and PC2 are connected to a common Ethernet hub to form the first LAN segment. PC1 is connected to PC3 with the HPNA adapter to form a second LAN segment, and PC1 is connected to PC4 with the wireless adapter to form a third LAN segment. The bridge can be used to connect the Ethernet network adapter, the HPNA network adapter, and the wireless network adapter on PC1, to forward traffic from one LAN segment to another and enable all of the computers to communicate with each other.

Without a network bridge (or additional routing configurations or bridging hardware), only PC1 can communicate with each of the other computers because it is the only computer that has connections to all three LAN segments. Because PC2, PC3, and PC4 use different types of network media, they are on different LAN segments, and they can only communicate with PC1.

(3) Network bridge types

There are two important types of network bridges, the LAN bridge and the Ethernet bridge, and they can be wired or wireless.

(a) LAN bridges

A LAN bridge connects two or more LANs at the second layer of the OSI model, receiving packets from a LAN segment connected to one port and forwarding them to another connected to a different port. They employ varying mechanisms. A simple LAN bridge regulates the transmission of frames to avoid congestion on the network. A learning LAN bridge remembers (learns) the Ethernet address of each frame it receives, in order to record which devices are connected to each port. It can then examine the destination address of each received frame to determine whether or not it should be forwarded to another part of the network. This selective forwarding improves the efficiency of communications across the network.

(b) Ethernet bridges

Ethernet bridges were originally used to connect homogeneous networks and forward packets between them. They allowed users to make the most of their network's size while not exceeding the maximum wire length, number of repeaters, or attached devices. The use of Ethernet bridges has expanded to include bridges between different networks connecting multiple networks together. There are two types of Ethernet bridge: remote and local. Local Ethernet bridges connect multiple LAN segments directly. Remote bridges connect multiple LAN segments in different areas, usually across telecommunication lines. The latter have an Ethernet interface on one side and a serial interface on the other. A similar device is used to connect to the other side of the serial line. These two types of bridges are most commonly used in WAN links, where network cable installation is not possible.

Additionally, a wireless Ethernet bridge converts a wired Ethernet device for use on a wireless computer network. Wireless Ethernet bridges and USB adapters are both sometimes called wireless media adapters as they enable devices for utilizing Ethernet or USB physical media. Wireless Ethernet bridges support game consoles, digital video recorders and other Ethernet-based consumer devices as well as ordinary computers.

11.3.2 Network gateways

A network gateway is a device with dedicated hardware and software that translates between two different protocols, making communication possible between networks of different architectures and protocols. The job of a gateway is much more complex than that of a network router or switch due to this conversion function.

Gateways are indispensable for communication between terminals connected to heterogeneous networks using different protocols and having different network characteristics. They provide the connectivity between systems at remote locations with the target system to enable different network applications. Network gateways are also called protocol converters or translators, and can operate at any layer of the OSI model.

In industrial control networks, gateways are designed for system integrators, machine builders and end users. Figure 11.11 gives one example, where a SCADA gateway works as a data concentrator.

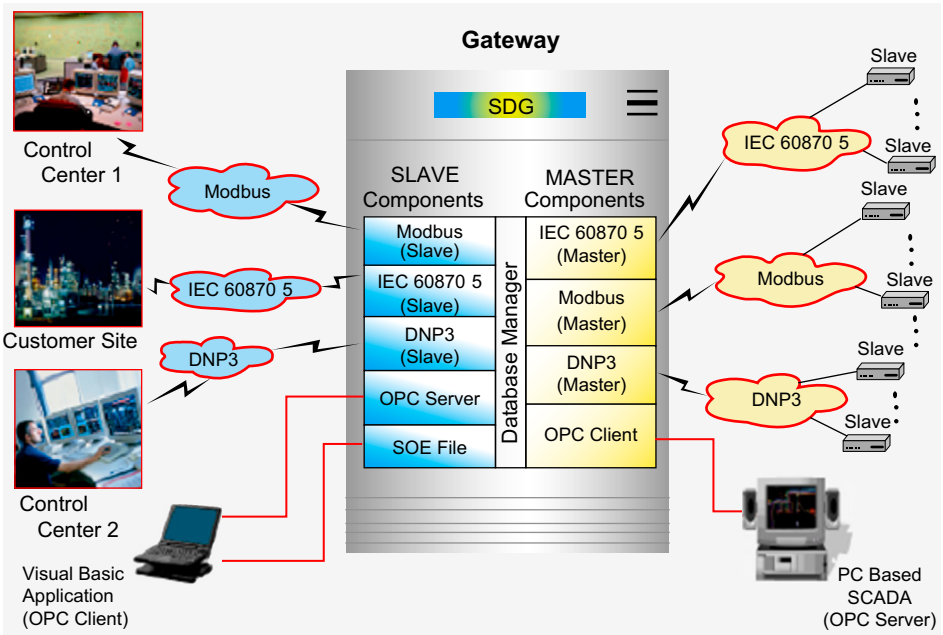


FIGURE 11.11

A SCADA data gateway for collecting data from OPC Server, IEC 60870-5, DNP3, or Modbus slave devices and then supplying the data to other control systems supporting OPC Client, IEC 60870-5, DNP3, and or Modbus communication protocols.

This SCADA data gateway collects data from an OPC Server, IEC 60870-5, DNP3, or Modbus slave devices and then supplies the collected data to other control systems supporting OPC Client, IEC 60870-5, DNP3, and/or Modbus communication protocols.

(1) Functions and categories of network gateways

Network gateways have been used for so many different functions, so that defining a gateway is no longer a simple task. In industrial applications, there are three main categories of gateways: protocol gateways; application gateways; and security gateways. The first two use data tunneling, resolve frame format mismatches, and resolve transmission rate (speed) mismatches. Security gateways are actually firewalls that use packet filtering, IP address checking, user identifier verification, and so on.

Data tunneling. Tunneling is a relatively common technique for passing data through an otherwise incompatible network region. Data packets are encapsulated with framing that is recognized by the network that will be transporting it. The original framing and formatting are retained, but are treated as data. When the packet reaches its destination, the host unwraps it and discards the wrapper, restoring the packet to its original format. Thus unacceptable packets are disguised in an acceptable format. Quite simply, tunneling can be used to defeat firewalls by encapsulating protocols that would otherwise be blocked from entry to a private network region.

Resolve frame format mismatches. Different networks have frame structures and media access mechanisms that make them unable to interoperate directly. The current generation of multiple protocol switches usually provides a high-bandwidth backplane that functions as a layer 2 protocol converting gateway which provides on-the-fly translation of the dissimilar portions of the frame structures.

Resolve transmission rate mismatches. The memory buffers in network devices can resolve the transmission rate differences. This memory buffers incoming and outgoing packets long enough for the gateways to determine which, if any, of the access list permissions apply. It can also be used to buffer the speed mismatches that may exist between the various network technologies.

(a) Protocol gateways

Protocol gateways usually convert (or translate) between protocols. Figure 11.12 gives an example of protocol gateways used in industrial networks. Transporting data between RS-232/422/485 and ASCII protocols up to a Fieldbus/Ethernet network is done by a serial gateway, often referred to as

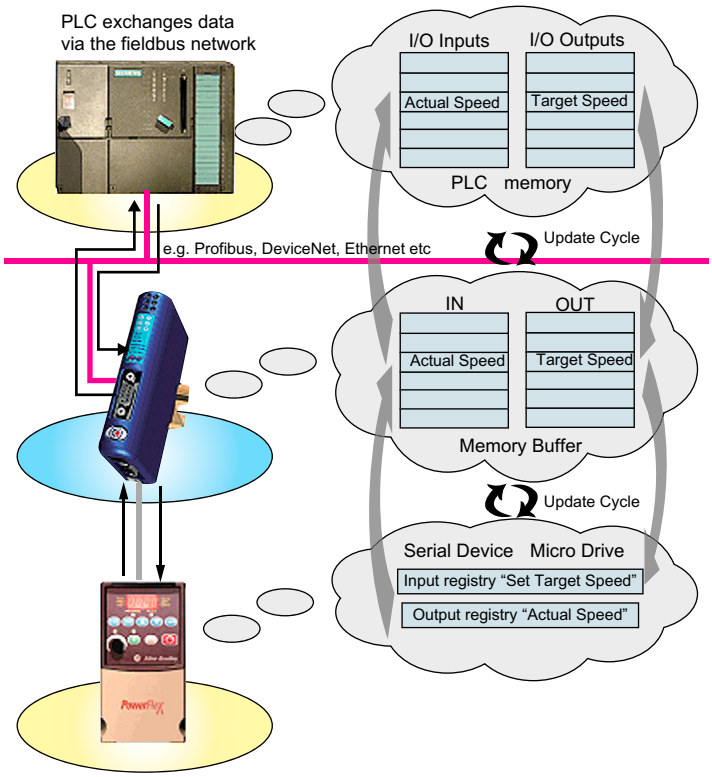


FIGURE 11.12

An example of protocol gateways used in industrial control networks: a serial gateway can collect data from the devices on the serial RS232/422/485/ASCII subnetwork, where the data are stored in the memory, which can be accessed by the Fieldbus/Ethernet network.

a protocol converter. The Anybus Communicator, for example, does this conversion by using temporary memory storage inside the gateway. The data from the serial device are mapped into a local memory in the gateway. With configuration software, the serial gateway can define which data bytes are from the serial data stream and what bytes are just command and control information. Advanced modes of, for example, the Anybus Communicator can also act as an active gateway controlling and communicating with several serial devices (multiple-drop function) using a common protocol such as Modbus RTU and linking selected data to an industrial network such as Profibus or Ethernet. It can also be used as an Internet or website enabler to give serial devices a website-based management, monitoring and control function.

(b) Application gateways

The typical application gateway accepts inputs in one format, translates it, and ships the outputs in a new format. The input and output interfaces can either be separate, or use the same network connection. A single application can have multiple application gateways. For example, electronic mail can be implemented in a wide variety of formats. Servers that provide electronic mail may be required to interact with other electronic mail servers, regardless of their format. The only way to do this is to support multiple gateway interfaces.

Application gateways can be used to connect LAN clients to external data sources. This type of gateway provides for local connectivity to an interactive application. Keeping the logic and executable code of the applications on the same LAN as the client base avoids lower bandwidth, higher-latency WANs, which in turn, enables better response time to clients. The application gateway would then ship an I/O request to the appropriate networked computer that housed the data requested by the client. The data are fetched and reformatted as needed for presentation to the client.

Using a gateway to fit between Ethernet and other protocols such as DeviceNet, Profibus or simple serial protocols also allows the gateway to serve data from that network directly to a built-in website server, enabling true web-based control and monitoring. The area of converting data from a device with a serial port to Ethernet, or any other network requires a more detailed explanation. The fact that the serial port may not have a layer 7 protocol at all makes the data linking difficult. An example is a device such as a barcode reader just transmitting ASCII data embedded in a device-specific serial data frame.

(c) Security gateways

The security gateway is a network gateway that acts as a firewall to filter packets and separate a proprietary network, such as an intranet, from a public one, such as the Internet. Network firewalls provide important safeguards for any network connected to the Internet. Firewalls are configured and managed to enforce a security policy tailored to particular needs of a given company or entity. To protect a local network, a gateway is usually used to connect between the local and public network. Through the use of network address translation techniques, a gateway provides security as a firewall while enabling machines with local IP addresses to access the Internet through the unique global address of the gateway. The gateway will receive the packet from the local machine, and substitute its external IP address and a new port address into the source fields of the IP and UDP headers. Data packets entering or exiting a local network are screened or filtered in a firewall. A gateway may also secure data packets transmitted between, for example, certain local networks, so acting as both a firewall and a VPN (virtual private network) gateway.

In the network for an enterprise, a gateway computer is often also acting as a proxy server and a firewall server. A gateway is often associated with both a router, which knows where to direct a given packet of data that arrives at the gateway, and a switch, which furnishes the actual path in and out of the gateway.

(2) Basic components of a network gateway

Similar to network routers, network gateways ordinary computers (PC) with special functionality. Gateways and computers both have a microprocessor, a motherboard, RAM, flash, and the interface boards as I/O ports. The main difference between them is that a network gateway embeds application software to perform special tasks to convert between network protocols, match different network data rates, and/or to control network traffic between two or more networks, etc. There are seven major internal components of a gateway: CPU, RAM, NVRAM, Flash, ROM, console, and interfaces. A photograph and a block diagram of a gateway system board are shown in Figure 11.5.

With the operating system included, a gateway may consist of software components such as protocol translators, data tunneling handlers, impedance matching devices, fault isolators or fault recorders, signal translators, and/or web servers as necessary to provide system interoperability. In addition to these components, gateways always need a data rate matcher and serial-to-fieldbus converters because gateways are slower than bridges, switches, and (non-gateway) routers.

Common features for network gateways include stackability, rack mounting, LED indicators, integrated firewall, and VPN. In a gateway, an integrated firewall is for security. VPN is a connection that has the appearance and many of the advantages of a dedicated link but occurs over a shared network. Using data tunneling, data packets are transmitted across a public routed network in a private "tunnel" that simulates a point-to-point connection and allows network protocols to traverse incompatible infrastructures.

A computer running Microsoft Windows, however, describes this standard networking feature as Internet Connection Sharing, which will act as a gateway, offering a connection between the Internet and an internal network. Such a system might also act as a server. The Dynamic Host Configuration Protocol (DHCP) is a protocol used by networked devices (clients) to obtain various parameters necessary for the clients to operate on the Internet.

(3) Specifications and configurations of network gateways

The general specifications for network gateways include: power; LED for status indication; communication ports with the interface devices such as RS232 / RS422 / RS485, and so on; isolation between communication ports and power supply; I/O data rate such as a 100-word input and output; bus address between 0 and 255 through setup software, etc.

The following example introduces the gateway configuration. The gateway is presumed to have two sides: the WAN side connects to a cable modem and the LAN side connects to a private network via a hub or switch. Its main function is to route the traffic from the computer to the Internet and vice versa. A computer with two network interface cards can act as a gateway. It routes the network traffic between two logically and physically different networks.

In this case, the public side of the gateway is configured and the IP address, which is assigned by the ISP (internet service provider) is assigned. DNS (domain name system) server, subnet mask, ISP gateway IP address and host name will also be set. Additionally, if the ISP uses PPPoE (Point to Point

Protocol over Ethernet) this must also be enabled. To configure the private side, it has to enable DHCP, which will automatically set the parameters required for the computer to be a part of the network.

The last step in the configuration is to configure each PC in such a way that it automatically gets all the settings from the DHCP server. The TCP/IP protocol needs to be properly installed in each computer in the network.

Each PC in the network then needs rebooting, after which there should be a LED blinking underneath the network icon on the right side of the task bar. If everything is done, the internet can be then accessed, enabling sharing of the printer and data in the network.

The firewall can also be configured at the same time to police for unauthorized network traffic.

11.3.3 Network repeaters

Network repeaters are electronic devices used to expand the boundaries of a wired or wireless network by regenerating the incoming signals to preserve signal integrity, and to extend the distance over which data can travel. A repeater typically contains a number of ports which connect between network nodes. They often connect to other network repeaters through a common data channel such as an expansion port, thus allowing communication between networks connected to different repeaters. Network repeaters work at the physical layer; layer 1 of the OSI network model.

A network repeater regenerates the received signals and then retransmits them on other segments (Figure 11.13). To pass data through the repeater in a usable fashion, the protocols must be the same on each segment. This means that a repeater will not enable communication, for example, between an IEEE 802.3 segment (Ethernet) and an IEEE 802.5 segment (token ring), because they cannot translate an Ethernet packet into a token ring packet.

However, using repeaters does not influence the real-time behavior of a system because in terms of transmission behavior it corresponds to a network that consists only of lines.

(1) Functions and types of network repeaters

Network repeaters use several different connector types. Industrial repeaters belong to one of three categories; industrial network repeaters, Fieldbus repeaters, and serial interface repeaters. Of course, these repeaters can either be wired or wireless. Figure 11.14 illustrates some of the industrial applications of network repeaters.

(a) Industrial network repeaters

Industrial networks can be CAN, SCADA, Ethernet or LAN, as discussed in Chapter 10. However, network repeaters are also important components of larger CAN networks, as CAN repeaters. These

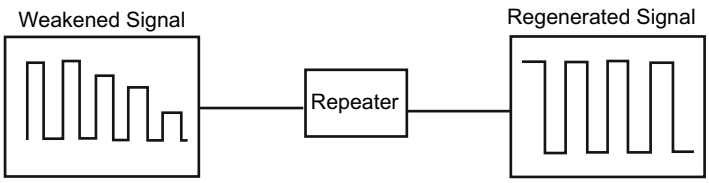


FIGURE 11.13

Network repeater enables signals to travel longer distances over a network by regenerating the signals.

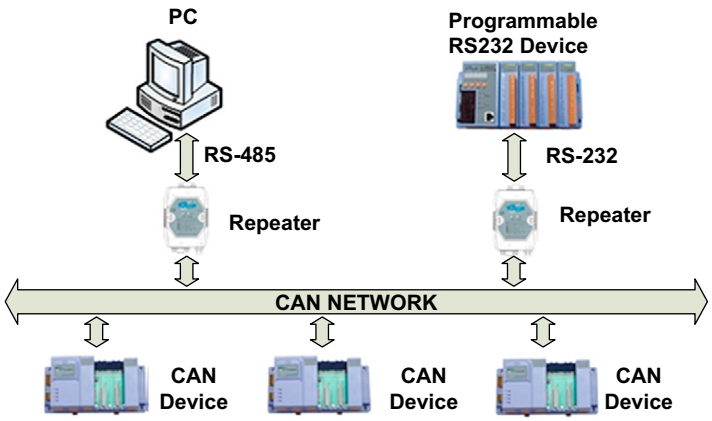


FIGURE 11.14

Some examples of industrial repeaters used in industrial networks.

support star or tree networks and establish a physical coupling of two or more segments of a CAN bus network. They can also be used to refresh signal shapes in networks with high node counts; and also for coupling of different physical CAN layers by means of high- or low-speed repeaters, optical repeaters, or optical star couplers.

(b) Fieldbus repeaters

As cable lengths increase, control signals deteriorate and can become unreadable by devices on Fieldbus networks. Fieldbus repeaters can be used to such avoid deterioration by taking an incoming signal from one network segment, cleaning it of errors, and re-transmitting to another network segment, maintaining signal quality and message integrity. This function is performed in a bi-directional fashion to support the network. They are fully compliant with the physical layer definition and specifications for Foundation Fieldbus, Profibus, and Interbus, etc. For example, if a Fieldbus network with direct current power can support a network segment cable length of up to about 1900 meters, up to four repeaters may be used in series, providing for the total cable length of any particular network trunk or spur to be increased up to 5 times the cable length of one segment. This allows for a Fieldbus device to be located up to 9.5 kilometres (5.9 miles) away from a control room.

(c) Serial interface repeaters

In industrial systems, serial interface repeaters can be used for interfaces such as the RS-232, RS-422 or RS-485 devices. Serial RS-232, RS-422, and RS-485 ports send data over a single wire. The device boosts attenuated RS-232/422/485 signals to extend the network's operating distance up to a number of kilometres. The use of serial interface repeaters also increases the maximum number of connectable nodes to hundreds of device units. Serial interface repeaters can offer a range of intelligent features that simplify the implementation and use of these devices. By employing special circuitry, they are able to detect the data flow automatically and switch the direction of the data lines accordingly.

(2) Specifications and configurations of network repeaters

Technically, a network repeater can be: a chipset or an ASIC; a printed circuit board or an integrated circuit card; stand-alone or enclosed; full-duplex or half-duplex; wired or wireless; and have two or more ports. Selection requires an analysis of network protocols. Network protocols working for physical layers or field levels may be also prescribed in all or some of network repeaters. The network protocols for industrial repeaters may be for industrial networks including CAN, DeviceNet, ControlNet, Foundation Fieldbus, Profibus, and Interbus, etc.

In terms of the OSI model, network repeaters are OSI physical layer devices having function switches and routers do not. In wireless LAN (WLAN), access points serve as repeaters only when operating in repeater mode. Most of the repeaters for wired networks may not require configuration because they are physical layer devices. However, some wireless repeaters may need to be configured with regard to the bandwidth, speeds, etc. of the access points they serve.

Problems

1. Please explain why the functionality of network devices can be specified in accordance with the OSI model: why hubs and repeaters work at the physical layer; bridges and switches work at the data link layer; routers work at the network layer; gateways work at any layer.
2. Hubs are said to be network devices without intelligent networking functions because a frame is just "broadcast" to every one of its ports in a hub, and hence it has no way of distinguishing which port a frame should be sent to. However, one hub type is the intelligent hub. Please explain what the differences are between intelligent networking functions and hub intelligence.
3. It seems that both hubs and switches use the CSMA/CD protocol to deal with traffic collisions in network buses; however, a switch board has buffers and software managers which a hub does not have. Do you think switches are more powerful than hubs in dealing with traffic collisions on network buses?
4. It is possible to connect several hubs together in order to centralize a large number of machines; this is sometimes called a daisy chain. Can you plot a diagram of daisy chain topology?
5. Based on Figure 11.4, explain the role of routers in creating subnetworks, decreasing the size of collision domains, and increasing bandwidth.
6. Please explain your understanding of the differences between network routers and network switches based on their functions. Do you agree that a switch is just concerned with transmitting data packets, whereas a router routes packets to other networks until the packet reaches its destination?
7. Based on Figures 11.6, 11.7, 11.8 and 11.9, please explain how the routing tables and routing protocols can help the routers to determine the best path to a destination in a network.
8. Explain why RIP is one of the most commonly used interior routing protocols but is unsuitable for large networks. Also explain why other routing protocols such as IGRP, EIGRP, OSPF and IS IS offer other advantages.
9. Please explain why a distance vector routing protocol commonly uses hop count as the routing metric, and routing table updates occur frequently even when the network topology has not changed.
10. Explain the way that static routing is where routes are manually entered into a routing table; whereas in dynamic routing, routes are automatically discovered and recorded in routing tables.
11. Please read further on the spanning tree protocol, and then explain how it is used to generate a loop free network topology for the transparent bridging algorithm used in network bridges.
12. What techniques can help Protocol and application gateways to perform their functionality, and why?
13. Please try to list all the differences between the network devices hub, switch, router, bridge, gateway, and repeater.
14. Is the firewall in your PC or laptop a security gateway?

Further Reading

Electronics Manufacturers (http:// www.electronics manufacturers.com). Networking equipment. http://www.electronics manufacturers.com/info/networking equipment/. Accessed: October 2008.

L com (http://www.l com.com). Datacommunications tutorial. http://www.l com.com/content/Article.aspx?Type=L&ID=208. Accessed: October 2008.

Wikipedia (http://en.wikipedia.org). Ethernet hub. http://en.wikipedia.org/wiki/Ethernet hub. Accessed: October 2008.

Wikipedia (http://en.wikipedia.org). Network switch: ethernet hub. http://en.wikipedia.org/wiki/Network switch. Accessed: October2008.

DUX Computer Digest (http://www.duxcw.com/index.html). Forums on Networking equipment: http://www.duxcw.com/yabbse/index.php. Accessed: October 2008.

Buzzle (http://www.buzzle.com). Computer networking articles. http://www.buzzle.com/articles/computer networking/. Accessed: October 2008.

Cisco (http://www.cisco.com). Networking, routers and routing. http://units.folder101.com/cisco/sem2/Notes/ch6 routing/routing.htm. Accessed: October 2008.

Cisco (http://www.cisco.com). Ciso router hardware, software and cofiguration. http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/netsys/netsysug/routconf.htm. Accessed: October 2008.

Skullbox.Net (http://skullbox.net). Routers and routing articles. http://skullbox.net/articles.php. Accessed: October 2008.

Silica Valley (http://www.silicavalley.com). Networking equipments. http://www.electronics manufacturers.com/info/networking equipment/. Accessed: October 2008.

Wikipedia (http://en.wikipedia.org). Router. http://en.wikipedia.org/wiki/Router. Accessed: October 2008.

Wikipedia (http://en.wikipedia.org). Control plane. http://en.wikipedia.org/wiki/Control Plane. Forwarding plane. http://en.wikipedia.org/wiki/Forwarding Plane. Accessed: October 2008.

Wikipedia (http://en.wikipedia.org). Network bridge. http://en.wikipedia.org/wiki/Bridging (networking). Accessed: October 2008.

GlobalSpec (http://www.globalspec.com). Networking equipments. http://communication equipment.globalspec.com/ProductFinder/Communications Networking/Networking Equipment. Accessed: October 2008.

Networking Tutorial (http://www.networktutorials.info). network Communication devices. http://www.networktutorials.info/communication devices.html. Accessed: October 2008.

Answers.com (http://www.answers.com). Computer Desktop Encyclopedia router: http://www.answers.com/topic/router. Accessed: October 2008.

Webopedia (http://www.webopedia.com). Differences between hub, switches, and routers. http://www.webopedia.com/TERM/Differences.htm. Accessed: October 2008.

Kioskea (http://en.kioskea.net). Networking equipment bridges. http://en.kioskea.net/lan/ponts.php3. Accessed: October 2008.

HIRSCHMANN (http://www.hicomcenter.com). Field network and Fieldbus devices. http://www.hicomcenter.com/Deutsch/Training/index.phtml. Accessed: October 2008.

Secure Computing (http://www.securecomputing.com). Security gateways. http://www.securecomputing.com/index.cfm?skey=20. Accessed: October 2008.

CHAPTER

Field interfaces 12

As mentioned in Chapter 3, all industrial control systems can usually be structured into a set of hierarchical levels. Figure 3.1 gives a definition of such a set; the management level, the control level, the field level, and the equipment level. Similarly to computer networks, communication can take place on several hierarchical levels, implemented by means of special devices termed interfaces.

In this definition, the network of field devices such as sensors and actuators is defined as the field level. In fact, the field level includes the systems which connect the field devices with the programmable controllers in order to permit communication between them. Communication at field level are obviously necessary to real-time control and automation because the process states and device statuses must be made available to the various, often remotely distributed, programmable controllers and visualization stations. Safe and reliable communications at the field level are therefore essential.

There are three types of interfaces offering real-time process control and production automation: the Actuator Sensor Interface (AS-Interface, or AS-i), the highway addressable remote transducer (HART), and fieldbuses.

HART communication uses conventional 4 to 20 mA current loop for data transmission, providing a very simple point-to-point connection between operating and field devices without needing additional wiring. The HART protocol is therefore useful when smart field devices, such as intelligent and programmable sensors and actuators, are to be integrated into an already existing plant. With the appropriate instrumentation, however, HART is also suitable as a communication system for extended plants. The only prerequisite is that the field devices are connected according to the conventional 4 to 20 mA technique.

Fieldbuses are wired completely differently, replacing the analog 4 to 20 mA current loops with a simple two-wire line, running from the control station to the field connecting all devices in parallel. Information transmission is purely digital, including control and process monitoring information as well as the commands and parameters required for start-up, device calibration and diagnosis.

When only binary (on/off) states need to be transmitted, the relevant system components can be networked via a simplified bus system. For applications in hazardous areas, the open bus system AS-Interface is a good solution. If required, this can be integrated via a special connection into more powerful fieldbus systems.

Apart from the above, two other solutions are avilable, adopting a middle course. In both cases, the field devices are wired according to the conventional 4 to 20 mA technique, but the lines are not run up to the control station because the signals are digitized before being supplied to a bus system. This task is accomplished by the field multiplexer. When the digital-analog conversion takes place in the control room, the system is called rackbus, whereas conversion in the field is performed by a remote I/O system.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10012-9

12.1 ACTUATOR–SENSOR INTERFACE

Industrial automation systems require large numbers of control devices, often including a number of binary (on/off) actuators and sensors. Conventional input and output (I/O) methods for wiring include point-to-point connections or bus systems. Typical batching valve wiring networks attach each of the I/O points to a central location, resulting in multiple-wire runs for each field device, which is expensive, and requires a considerable amount of space. These methods can prove to be too complex for networking simple binary devices and too slow for the interactions between the controllers and the controlled devices. Point-to-point wiring is the most common method of wiring in industry, but large wire bundles take up valuable space, installation is time-consuming, and troubleshooting is complex.

The Actuator Sensor Interface, or AS-Interface or AS-i, was developed by a group of sensor manufacturers and introduced into the market in 1994. Since that time, it has become the standard for discrete sensors and actuators in process industries around the world. It is also a bus system, used for low-level field applications in industrial automation to communicate with small binary sensors and actuators using the AS-Interface standard. This modernizes automation systems effectively and eliminates wire bundles completely, with only one wire cable required for all devices, compared to one cable from each device needed for point-to-point wiring. Junction boxes are also eliminated, and the size of the control cabinet needed is significantly reduced. Plug-and-play wiring supports all typologies. Figure 12.1 gives the locations of the AS-Interface in an industrial control network.

When compared with conventional I/O wiring methods, the AS-Interface has many advantages. The most important ones are:

1. Minimum wiring giving cost-savings. The AS-Interface needs a single cable, which uses simple serial connection to the controller, instead of parallel with a multitude of cables.
2. Fast and safe installation. Sensors and actuators are simply installed with modules on the AS-Interface cable. Contact pins in the modules penetrate the insulation of the cable and establish contact with the copper wire. Incorrect connections are practically impossible because of the design of the cable and the special piercing method.
3. Flexible configuration. Owing to the distributed and modular design, plant sections can be tested in parallel even before the overall solution is finished. This permits flexible modification and expansion.
4. Open system. AS-Interface is an open system, which means that it is independent of manufacturer and future-proof.

12.1.1 Architectures and components

There are two types of AS-Interface architectures as displayed in Figure 12.1.

(1) AS-Interface architecture: type 1

In the first type of AS-Interface architecture, a programmable controller such as a PLC, SCADA, or PC controls sensors and actuators via field-level buses, including Foundation Fieldbus, Profibus, etc.

As displayed in Figure 12.2, the AS-Interface has gateways directly connected to the field-level bus and the I/O module. The latter is the device which contacts the sensors and the actuators. A field-level bus may be able to support several AS-Interface gateways depending on the manufacturing specification and the system design, each of which fits a segment of an industrial control system.

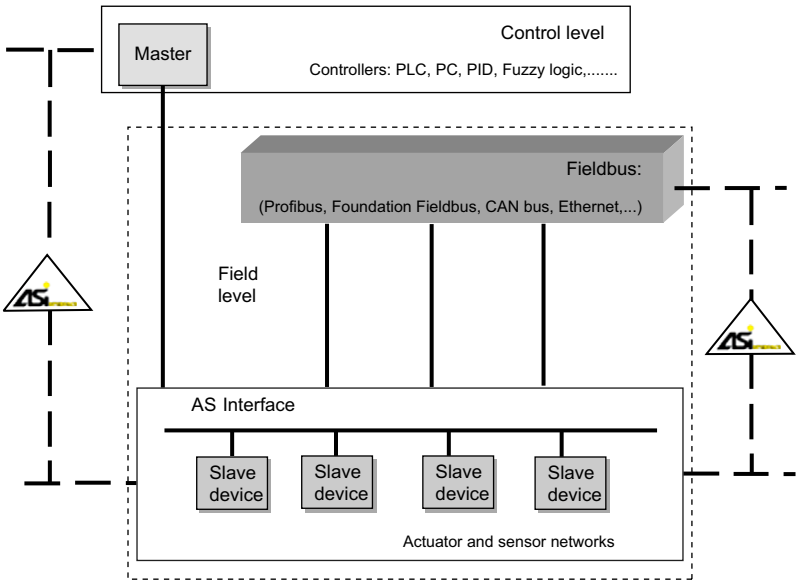


FIGURE 12.1

The functionality of Actuator Sensor Interface in industrial control systems or networks. The Actuator Sensor Interface can be at two locations in an industrial control systems or networks: between the controllers and the actuator/sensor networks and between the fieldbuses and actuator/sensor networks.

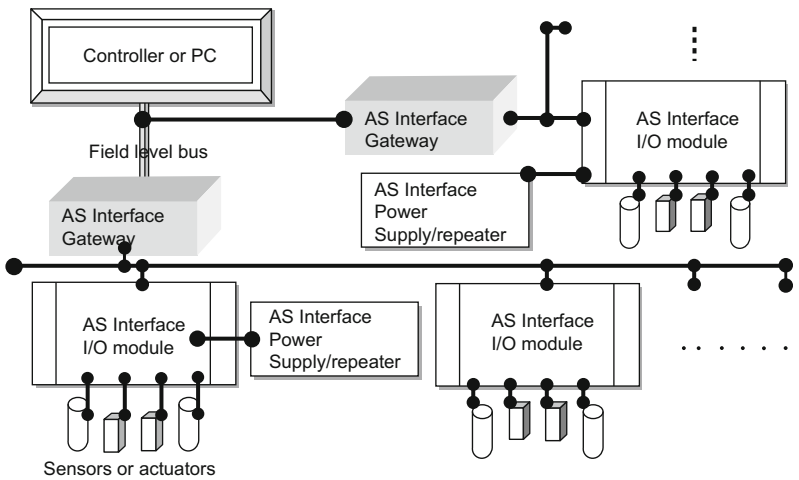


FIGURE 12.2

Actuator Sensor Interface architecture: type 1.

In this type of architecture, the AS-Interface requires the following components:

(1) Gateways

Gateways are devices that interface between the AS-Interface and a higher-level bus system. They are used when more complex applications are to be implemented using standard products. AS-Interface gateways are the core of the wiring system, handling the complete data transfer, by cyclically polling all participants (master/slave) connected to the wiring system. They can be placed anywhere in the AS-Interface segments. At present, one gateway routinely handles 124 inputs and 124 outputs over 31 addressable I/O modules, and setup is accomplished through the setup tools of the respective system.

(2) I/O modules

I/O modules are located between standard sensors and actuators and the AS-Interface. They are available for many kinds of application, including flat modules for limited-space applications, compact modules for a variety of mounting options, field modules that use cord grips instead of quick disconnects, and standard modules that use both the mechanically keyed AS-Interface cable and the standard 16 AWG round cable. For enclosures and junction boxes, enclosure modules connect the AS-Interface bus to a power rail system, and junction box modules for use within junction boxes.

(3) Power supplies and repeaters

In an AS-Interface, one single cable transmits both power and data. Power supplies contain internal data separation coils so that the capacitive filtering of the supply does not interfere with the data stream. Adding to the high interference immunity of AS-Interface is the power supply data isolation coil between the voltage transformer and the output so that the data signals are isolated from line noise. Repeaters extend AS-Interface networks by up to 100 meters; by using two in series an AS-Interface network can be up to 300 meters long. Repeaters do not require a network address and allow I/O modules to be placed anywhere along the network.

(4) AS-Interface safety at work

This extension of the AS-Interface allows safety equipment to be wired together on a two-wire cable rather than hardwired back to a panel. A maximum of 31 category 4 inputs, such as E-Stops, can be put on one cable. The parts included in this section are safety slaves, safety monitor, and configuration software.

In many applications, safety-relevant functions are a prerequisite. They take the form of emergency stop buttons near process lines, or the installation of safe sensors (e.g., safe photo grits and locking of safety-related doors) to automatically stop machines.

Integration of the safety requirement into the AS-Interface line through the terminology “AS-Interface safety at work”, can drastically reduce additional costs. The concept refers to connection of the safety-related switches by a safe AS-Interface module. There is also a safety monitor on the AS-Interface line that permanently observes the communication. The communication happens through a given and predetermined pattern in a dynamic code table with an 8 times 4 bits sequence. The safe monitor continuously compares “must” and “actual” values of the communication. In the case of the bit sequence 0000, the safe monitor switches off the safe relay in less than 40 milliseconds. Several safe monitors can be operated in one AS-Interface line arranged in any position.

Clear benefits of AS-Interface safety at work include the following: only one AS-Interface line is required for the communication of safe and non-safe data; full compatibility with all standard

AS-Interface devices; no specific communication mechanisms required; mixed applications on one and the same AS-Interface line possible; diagnosis of the safe modules via the standard AS-Interface master possible.

(5) AS-Interface encoders

In order to be able to meet the real-time requirements of many applications, a multiple-slave solution can be adopted. The position value, up to 16 (or 32) bits in length, is transferred to the gateways within a single cycle, via the four integrated AS-Interface chips used for control purposes. AS-Interface rotary encoders include 13-bit-Singleturn and 16-bit-Multiturn.

(6) Accessories

To complete the AS-Interface and to make the installation as easy as possible, various accessories such as hand-held addressing devices, mounting bases, simulators for higher-level bus systems, and sealing for flat cable and adaptor to round cable are available.

(2) AS-Interface architecture: type 2

In the second type of AS interface architecture, as shown in Figure 12.3, the AS-Interface master module resides inside a programmable controller such as a PLC, SCADA, or PC. The AS-Interface master terminal enables the direct connection of either analog AS-Interface slaves or digital. It does not manage the sensors and actuators via the field-level buses, but rather via the AS-Interface slave modules or cables. The modules are connected to each other by means of the AS-Interface cable, which can be branched with the cable branch device. A group of slave modules frames a segment of an

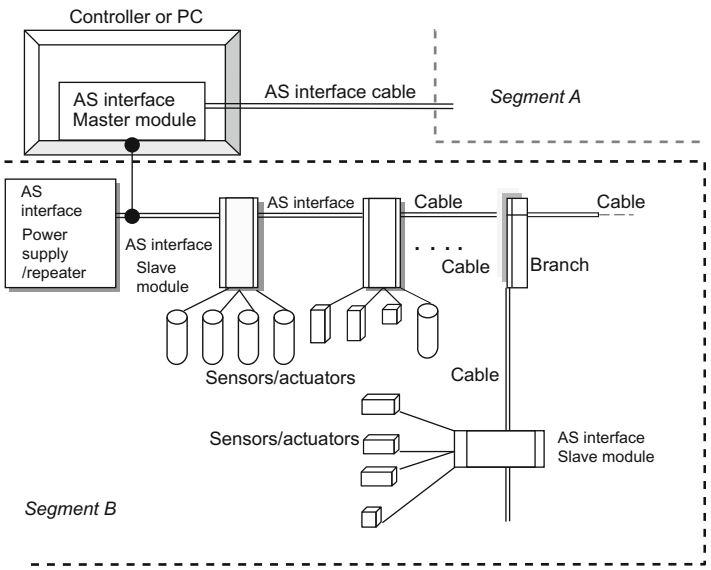


FIGURE 12.3

Actuator Sensor Interface architecture: type 2.

industrial control network with one interface cable. The AS-Interface master module is able to support several segments depending on the designed system capabilities.

In this type of architecture, AS-Interface requires the following components.

(1) AS-Interface masters

The AS-Interface master automatically controls all communication over the AS-Interface cable without the need for special software. The master can connect the system to a programmable controller such as a PLC, SCADA, or PC, act as a standalone controller, or serve as a gateway to higher-level bus systems. The following AS interface masters are currently available:

(a) Standard AS-Interface master. Up to 31 standard slaves or slaves with extended addressing mode can be attached to standard AS-Interface masters.

(b) Extended AS-Interface masters. They support 31 addresses that can be used for standard slaves or those with extended addressing mode. AS-Interface slaves with extended addressing mode can be connected in pairs (programmed as A or B slaves) to an extended AS-Interface master and can use the same address. This increases the number of addressable AS-Interface slaves to a maximum of 62. Due to this address expansion, the number of binary outputs is reduced to three per AS-Interface slave.

(2) AS interface slaves

All the nodes that can be addressed by an AS-Interface master are defined as AS-Interface slaves.

(a) AS-Interface slave assembly system. The following assembly systems are available:

- **(i)** AS-Interface modules. AS-Interface modules are AS-Interface slaves to which up to four conventional sensors and up to four conventional actuators can be connected. The standard coupling module, which is the lower section of a standard device, connects the user module to the yellow AS-Interface cable. The user module connects the sensors and actuators, while the application modules connect via screw terminals or connectors. Sensors and actuators with a built-in AS-Interface chip can be directly connected to the AS-Interface cable.
- **(ii)** Sensors/actuators with an integrated AS-Interface connection. Sensors/actuators with an integrated AS-Interface connection can be connected directly to the AS-Interface.

(b) Addressing modes are as follows:

- **(i)** Standard slaves. Standard slaves each occupy one address on the AS-Interface. Up to 31 standard slaves can be connected to the AS-Interface.
- **(ii)** Slaves with an extended addressing mode (A/B slaves). These can be operated in pairs at the same address with an extended AS-Interface master. This doubles the number of addressable AS-Interface slaves to 62.

One of these AS-Interface slaves must be programmed as an A slave using the addressing unit and the other as a B slave. Due to the address expansion, the number of binary outputs is reduced to three per AS-Interface slave. Slaves can also be operated with a standard AS-Interface master. For more detailed information about these functions, refer to the AS-Interface master discussion in the previous paragraphs.

(c) Analog slaves. Analog slaves are special AS-Interface standard slaves that exchange analog values with the AS-Interface master, and require special program sections to execute the sequential transfer of analog data. Analog slaves are intended for operation with extended AS-Interface masters. The

extended AS-Interface masters handle the exchange of analog data with these slaves automatically. No special drivers or function blocks are required in the user program.

(3) Further AS-Interface system components

Further AS-Interface components include the AS-Interface cable, the AS-Interface power supply unit, addressing unit, and SCOPE for AS-Interface.

(a) AS-Interface cable. The trapezoidal AS-Interface cable is recommended over standard two-wire round cable for quick and simple connection of slaves. It is available in different colors to signify its voltage rating, with color assignments as follows:
  (i) Yellow is used for data and control power between the master and its slaves.
  (ii) Black is the external output power cable up to 60 V DC.
  (iii) Red is the external output power cable up to 240 V AC.

The AS-Interface cable, designed as an unshielded two-wire cable, transfers signals and provides the power supply for the sensors and actuators connected using AS-Interface modules. Networking is not restricted to one type of cable. If necessary, appropriate modules or T pieces can be used to change to a simple two-wire cable.

(b) AS-Interface power supply unit. The AS-Interface power supply unit supplies power to the AS-Interface nodes connected to the AS-Interface cable. For actuators with particularly high power requirements, the connection of an additional power supply may be necessary (e.g., using special application modules). Data and control power are normally transmitted simultaneously via the AS-Interface cable. Power for the electronics and inputs is supplied by a special AS-Interface power supply that feeds a symmetrical supply voltage into the AS-Interface cable via a data-decoupling device.

(c) Addressing unit. The addressing unit allows simple programming of AS-Interface slave addresses.

(d) SCOPE for AS-Interface. SCOPE for AS-Interface is a monitoring program for Windows that can record and evaluate the data exchange in AS-Interface networks during the commissioning phase and during operation. SCOPE for AS-Interface can be operated on a PC under Windows in conjunction with an AS-Interface master communications processor.

12.1.2 Principles and mechanisms

The AS-Interface utilizes a single, trapezoidal, unshielded two-wire cable, which eliminates the extensive parallel control wiring required with most installations. In a network with the AS-Interface, a simple gateway interfaces the network into the field communication bus. Data and power are transferred over the two-wire network to each of the AS-Interface compatible field devices. The existing controller sees AS-Interface as remote I/O; therefore, AS-Interface connects to the existing network with minimal programming changes.

The AS-Interface system utilizes only one master per network to control the exchange of data. This allows the master to interrogate up to 31 slaves and update all I/O information within 5 ms (10 ms for 62 slaves). For slave connection, an insulated two-wire cable is recommended to prevent reversing polarity. The electrical connection is made using contacts that pierce the insulation of the cable, contacting the two wires, thus eliminating the need to strip the cable and wire to screw terminals. For

data exchange to occur, each slave must be programmed with an address that is stored internally in nonvolatile memory and remains even after power is removed.

(1) How the AS-Interface functions

The AS-Interface or AS-Interface system operates as outlined below:

1. Master-slave access techniques. The AS-Interface is a single-master system. This means that there is only one master per AS-Interface network to control the operations of process. This polls all AS-Interface slaves one after the other and waits for a response.
2. Electronic address setting. The address of an AS-Interface slave is its identifier, set only once within an AS-Interface system, either by using a special addressing unit or by an AS-Interface master. The address is always stored permanently on the AS-Interface slave.
3. Operating reliability and flexibility. The transmission technique used (current modulation) guarantees high operating reliability. The master monitors the voltage on the cable and the transferred data. It detects transmission errors and the failure of slaves and sends a message to the controller. The user can then react to this message. Replacing or adding AS-Interface slaves during normal operation does not affect communication with other AS-Interface slaves.

(2) Master-slave principle

The AS-Interface operates on the master-slave principle as discussed earlier. This means that the AS-Interface master connected to the AS-Interface cable controls the data exchange with the slaves via the interface to the cable of the AS-Interface.

Figure 12.4 illustrates the two interfaces of the AS-Interface master communication processor. The first interface is between the master CPU and the master communication processor; the second is between the master communication processor and AS-Interface cable. Process data and parameter assignment commands are transferred via the first interface, and user programs have suitable function calls and mechanisms available for reading and writing via this interface. On the other hand, information is exchanged with the AS-Interface slaves via the second interface between the master communication processor and AS-Interface cable.

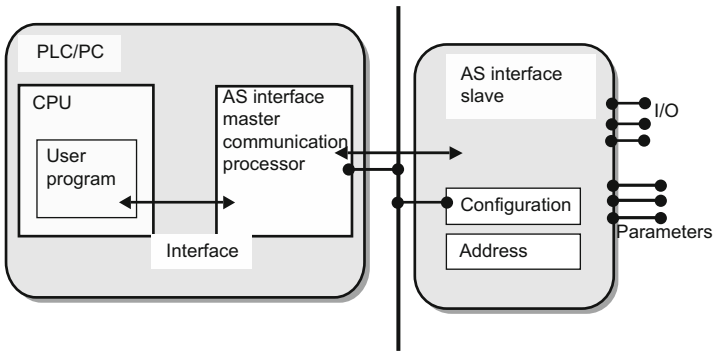


FIGURE 12.4

Actuator Sensor Interface operation.

(1) Tasks and functions of the AS-Interface master

The AS-Interface master specification distinguishes masters with different ranges of functions known as a profile. For standard AS-Interface masters and extended AS-Interface masters, there are three different master classes (M0, M1, M2 for standard masters, and M0e, M1e, M2e for extended masters). The AS-Interface specification stipulates the functions that a master in a particular class must be able to perform. The profiles have the following practical significance:

(a) Master profile M0/M0e. The AS-Interface master can exchange I/O data with the individual AS-Interface slaves. The station configuration on the cable, called the expected configuration, is used to configure the master.

(b) Master profile M1/M1e. This profile covers all the functions according to the AS-Interface master specification.

(c) Master profile M2/M2e. The functionality of this profile corresponds to master profile M0/M0e, but the AS-Interface master can also assign parameters to the slaves. The essential difference between extended AS-Interface masters and standard AS-Interface masters is that they support the attachment of up to 62 AS-Interface slaves using the extended addressing mode. Extended AS-Interface masters also provide particularly simple access for AS-Interface analog slaves complying with profile specifications.

(2) How an AS-Interface slave functions

The following describes some slave functions. For more detailed information on the ID codes, refer to the manufacturer's description.

(a) Connecting to the AS-Interface cable. The slave has an integrated circuit (AS-Interface chip) that provides the attachment of an AS-Interface device (sensor/actuator) to the common bus cable to the master. The integrated circuit contains four configurable data inputs and outputs, and four parameter outputs. The operating parameters, configuration data with I/O assignment, identification code, and slave address are stored in additional memory (e.g., EEPROM).

(b) I/O data. Data that were transferred to the AS-Interface slave are available at the data outputs. Values at the data inputs are made available to the master when the slave is polled.

(c) Parameters. Using the parameter outputs of the slave, the AS-Interface master can transfer values that are not interpreted as simple data. These parameter values can be used to control and switch over between internal operating modes of the sensors or actuators. It could, for example, be possible to update a calibration value during the various operating phases. This function is possible with slaves that have an integrated AS-Interface connection providing they support the function in question.

(d) Configuration. The input/output configuration (I/O configuration) indicates which data lines of the AS-Interface slave are used as inputs, outputs, or as bidirectional outputs, and can be found in the description of the AS-Interface slave. In addition to the I/O configuration, the type of AS-Interface slave is specified by an identification code; newer AS-Interface slaves are identified by three identification codes (ID code, ID1 code, ID2 code).

(3) Data transfer

Data transfer is one of the most important functions in the AS-Interface operation, as explained below.

(1) Information and data structure

Before introducing the operating phases, and their functions, a brief outline of the information structure of the AS-Interface master/slave system is necessary. Figure 12.5 shows the data fields and lists of the system as configured in the system structure diagram that was given in Figure 12.4. The following structures are found on the AS-Interface master:

(a) Data images. These contain temporarily stored information:
- **(i)** actual parameters that are an image of the parameters currently on the AS-Interface slave;
- **(ii)** actual configuration data that contains the I/O configurations and ID codes of all connected AS-Interface slaves once these data have been read from the AS-Interface slaves;
- **(iii)** the list of detected AS interface slaves (LDS) that specifies which AS-Interface slaves were detected on the AS-Interface bus;
- **(iv)** the list of activated AS-Interface slaves (LAS) that specifies which AS-Interface slaves were activated by the AS-Interface master. I/O data are only exchanged with activated AS-Interface slaves.

(b) I/O data. The I/O data are the process input and output data.

(c) Configuration data. These are nonvolatile data (e.g., stored in an EEPROM), which remain unchanged even following a power failure.

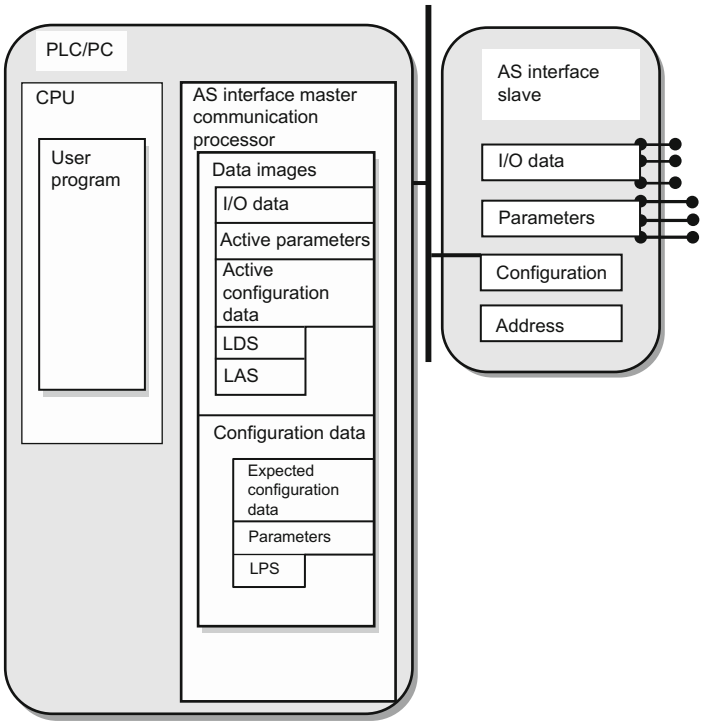


FIGURE 12.5

Data transfer between the master and the slave of an Actuator Sensor Interface.

(i) Expected configuration data are selectable comparison values which allow the configuration data of the detected AS-Interface slaves to be checked.

(ii) List of permanent AS-Interface slaves (LPS) that specifies which slaves are expected on the cable by the master. The AS-Interface master checks continuously whether all the slaves specified in the LPS exist, and whether their configuration data match what is expected. The AS-Interface slave has the following structures: (a) I/O data; (b) parameters; (c) actual configuration data; the configuration data include the I/O configuration and the ID codes of the AS-Interface slave; (d) address. Slaves have an address of 0 when installed, so to allow data exchange, their addresses must be set. The address 0 is reserved for special functions.

(2) The operating phases

Figure 12.6 illustrates the individual operating phases.

(a) Initialization mode. Also known as the offline phase, this sets the basic status of the master. The module is initialized after switching on the power supply or following a restart during operation. During the initialization, the images of all the slave inputs and the output data from the point of view of the application are set to 0 (inactive). After switching on the power supply, the configured parameters are copied to the relevant field so that subsequent activation uses the preset parameters. If the AS-Interface master is reinitialized during operation, the values from the parameters field that may have changed in the meantime are retained.

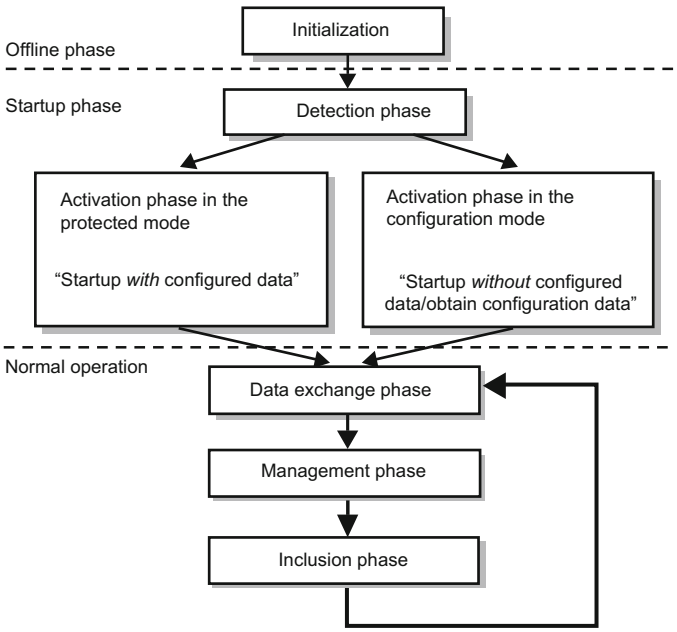


FIGURE 12.6

How the individual operating phases work in data transfer through an Actuator Sensor Interface.

(b) Start-up phase.

(i) Detection phase. During start-up or after a reset, the AS-Interface master runs through this phase, during which it detects which slaves are connected and what type they are from their configuration data. Configuration files contain the I/O assignment and the slave type (ID codes). The master enters detected slaves in the list of detected slaves (LDS).

(ii) Activation phase. After the slaves are detected, the master sends a special call which activates them. When activating individual slaves, a distinction is made between two modes on the AS-Interface master:

Master in the configuration mode: all detected stations (with the exception of the slave with address 0) are activated. In this mode, it is possible to read actual values and to store them for a configuration.

Master in the protected mode: only the stations corresponding to the expected configuration stored on the AS-Interface master are activated. If the actual configuration differs from this, the AS-Interface master indicates this.

The master enters activated AS-Interface slaves in the list of activated slaves (LAS).

(iii) Normal mode. On completion of the start-up phase, the AS-Interface master switches to the normal mode.

(iv) Data exchange phase. In the normal mode, the master sends cyclic data (output data) to the individual AS-Interface slaves and receives their acknowledgment messages (input data). If an error is detected during transmission, the master repeats the appropriate poll.

(v) Management phase. During this phase, all existing jobs of the control application are processed and sent. Possible jobs are, for example:

Parameter transfer: four parameter bits (three parameter bits with AS-Interface slaves with the extended addressing mode) are transferred to a slave and are used, for example, for a threshold value setting.

Changing slave addresses: this function allows the addresses of AS-Interface slaves to be changed by the master if the AS-Interface slave supports this particular function.

(vi) Inclusion phase. In the inclusion phase, newly added AS-Interface slaves are included in the list of detected AS-Interface slaves and, providing the configuration mode is selected, they are also activated (with the exception of slaves with address 0). If the master is in the protected mode, only the slaves stored in the expected configuration of the AS-Interface master are activated. With this mechanism, slaves that were temporarily out of service can be included again.

(3) Interface functions

Various functions are available on the interface to control the master and slave interaction from the user program. The possibilities are explained below. The possible operations and the direction of data flow are illustrated in Figure 12.7.

(a) Read/write. When writing, parameters are transferred to the slave and to the parameter images on the communication processor; when reading, parameters are transferred from the slave or from the communication processor parameter image to the CPU.

(b) Read and store (configured) configuration data. Configured parameters or data are read from the nonvolatile memory of the communication processor.

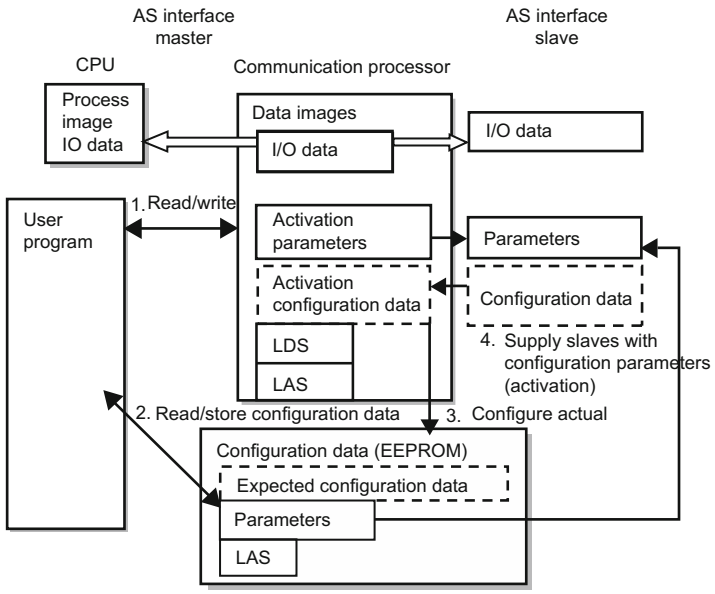


FIGURE 12.7

How an Actuator Sensor Interface performs an operation.

(c) Configure actual. When reading, the parameters and configuration data are read from the slave and stored permanently on the communication processor; when writing, the parameters and configuration data are stored permanently on the communication processor.

(d) Supply slaves with configured parameters. Configured parameters are transferred from the nonvolatile area of the communication processor to the slaves.

(4) Extended AS interface slaves with standard AS-Interface masters

The following information is about operating extended AS-Interface with standard AS-Interface masters.

(a) Slaves are connected to standard masters. The most significant slave bit (bit 4) of each A slave must be set to 0. The most significant parameter bit (bit 4) must also be set to 1 (default value). Without these settings, the A slave cannot be operated with a standard master.

(b) B slaves must not be connected to standard AS-Interface masters.

12.1.3 Systems and environments

As previously explained, the AS-Interface is an open, non-proprietary bus system. Both actuators and sensors are connected to programmable logic controllers using a non-shielded 2-wire cable on which the data and power are simultaneously transferred.

(1) AS-Interface system characteristics

The most important physical characteristics of the AS-Interface and its components are as follows:

1. Two-wire cable for data and power supply. A simple two-wire cable can be used, supplying both data and power. The power available depends on the supply unit used. For optimum wiring, mechanically coded AS-Interface cable is available, which prevents connections from being reversed and makes simple contact with the AS-Interface application modules using the penetration technique.
2. Tree structure network with a cable. The tree structure of the AS-Interface allows any point on to a cable section to be used as the start of a new branch.
3. Direct integration. Practically all the electronics required for a slave have been integrated on to a special integrated circuit, allowing the AS-Interface connector to be integrated directly in binary (on/off) actuators or sensors.
4. Increased functionality, more uses for the customer. Direct integration allows devices to be equipped with a wide range of functions. Four data and four parameter lines are available, so the resulting "intelligent" actuators/sensors have many possible uses; for example, monitoring, parameter assignment, wear or pollution checks, etc.
5. Additional power supply for higher power requirements. An external source of power can be provided for slaves with a higher power requirement.

(2) AS-Interface system limits

The AS-Interface system is currently subject to the limits below.

(1) Cycle time

(a) Maximum 5 ms with standard AS-Interface slaves.

(b) Maximum 10 ms with AS-Interface slaves that use the extended addressing mode. AS-Interface uses constant message lengths. Complicated procedures for controlling transmission and identifying message lengths or data formats are not required. This makes it possible for a master to poll all connected standard slaves within a maximum of 5 ms and to update the data on both the master and slave. If only one AS-Interface slave using the extended addressing mode is located at an address, this slave is polled at least every 5 ms. If two extended slaves (A and B slave) share an address, the maximum polling cycle is 10 ms. (B slaves can only be connected to extended masters.)

(2) Number of connectable AS-Interface slaves

(a) Maximum of 31 standard slaves.

(b) Maximum of 62 slaves in the extended addressing mode. AS-Interface slaves are the input and output channels of the AS-Interface system. They are only active when called by the AS-Interface master, triggering actions or transmit reactions when commanded. Each AS-Interface slave is identified by its own address (1 31). A maximum of 62 slaves using the extended addressing mode can be connected to an extended master. Pairs of slaves occupy one address; in other words, the addresses 1 31 can be assigned to two extended slaves per slot. If standard slaves are connected to an extended master, these occupy a complete address; in other words, a maximum of 31 standard slaves can be connected to an extended master.

(3) Number of inputs and outputs

(a) A maximum of 248 binary inputs and outputs is possible with standard modules.

(b) A maximum of 248 inputs and 186 outputs if the extended addressing mode is used. Each standard AS-Interface slave can receive and send 4 bits of data. Special modules allow each of these bits to be used for a binary actuator or a binary sensor meaning that an AS-Interface cable with standard AS-Interface slaves can have a maximum of 248 binary attachments (124 inputs and 124 outputs). All typical actuators or sensors can be connected in this way. The modules are used as distributed inputs/outputs. If modules with the extended addressing mode are used, a maximum of 3 inputs and 3 outputs is available per module; in other words a maximum of 248 inputs and 186 outputs can be operated with modules using the extended addressing mode.

(3) AS-Interface in a real-time environment

The system characteristics listed below means the AS-Interface can work in a real-time environment.

1. Optimized system for binary sensors and actuators and for simple analog elements.
2. Master-slave principle with cyclic polling.
3. Tree structure of the network.
4. Both data and power supplied by means of one unshielded two-wire cable.
5. Flat cable for contacting by piercing technology.
6. Modules act as remote I/O ports for conventional sensors and actuators.
7. Integrated slaves with their own AS-Interface capabilities.
8. No communication software in the slaves, only firmware in the self-configuring master.
9. Low costs, simple installation, easy handling, flexible networks, high reliability in an industrial environment, open and internationally accepted system that has many manufacturers and products.

There are three aspects of the AS-Interface that are of particular importance in real-time applications: connectivity, cycle time, and availability.

(1) Connectivity

AS-Interface has two distinct ways to connect to the first control level.

The first and most important is a direct connection like the type 2 AS-Interface architecture given in section 12.1.1. In this case, the system's master is an embedded part of a programmable controller such as a PLC, SCADA, or PC, running at its own cycle time. As the AS-Interface is an open system, a manufacturer of any kind of programmable controller can build a master for their own system; many are already available, with more under development.

The second way is to connect AS-Interface via a coupler to a higher fieldbus and to use it as a subsystem, which is the type 1 AS-Interface architecture explained in section 12.1.1. In this case, all data from the AS-Interface network are handled in one node of the fieldbus, and this is connected to the above-lying host together with other components of the higher fieldbus. The application program (user program) handles all data for the particular fieldbus. For real-time applications, an analysis of the cycle time and the availability of the combination of the two systems has to be done. The AS-Interface offers couplers to most known higher fieldbuses such as Profibus, CAN, etc., with others (e.g. LON, Fieldbus Foundation) under development. Together with its tree structure, AS-Interface thus offers the most flexible networking solution to any automation application.

(2) Cycle time

AS-Interface is a single-master system with cyclic polling. Thus, any slave is addressed in a definite time period.

For a complete network with 31 slaves, the cycle time is 5 ms; less with fewer slaves. (With very few slaves the cycle time can be shortened to less than 500 μs.) Analog data with more than 4 bits need several cycles, but do not affect the basic cycle time for binary sensors and actuators.

The cycle time includes all steps from and to the interface to the host system, and even includes one repetition. Data exchange with the host uses process I/O images at the end of each cycle stored in, for example, a dual-ported memory at the interface. Therefore no other steps are needed, for a direct connection to the control device.

This is asynchronous coupling, and in real-time applications this may present a restriction, but for many systems and applications this is short enough.

(3) Availability

Availability in this context means that a system will deliver reliable data and diagnostic values continuously and in time under all specified conditions, especially under severe electromagnetic noise. The answers to three questions are of special importance for real-time applications:

(a) Can electromagnetic noise or other faults disturb the reliability of data?
(b) How much time is necessary for the correction of a faulty transmission?
(c) How often does such a fault happen and can this affect the whole system?

12.2 HIGHWAY ADDRESSABLE REMOTE TRANSDUCER (HART)

HART is an acronym for highway addressable remote transducer, a two-way digital communication interface and protocol which works with 4 20 mA analog signaling. HART is chosen for traditional instrumentation equipment in industrial control systems, in most cases for the field level, and has therefore become one of the most popular industrial communication interfaces and protocols.

Historically, HART was the first implementation of a fieldbus that was developed in the early 1980s by a company named Rosemount Inc. for host management of field devices in process industries. In 1986, it was made an open protocol, then in July of 1993, the HART Communication Foundation was established to provide worldwide support for application of this technology. Nowadays, HART specifications continue to be updated to broaden the range of possible applications. A recent HART development, the device description language (DDL), provides a universal software interface to both new and existing devices.

12.2.1 HART communications

Most industrial control systems have numerous instruments and devices at the field level. While processes are running, the host controller therefore should communicate with these devices in order to (1) configure or reconfigure them, (2) gather diagnostics, (3) troubleshoot, (4) read additional measurements (5) determine device health and status, etc. A host in the system can either be a network server, a programmable controller, a hand-held communicator, or other device.

There are many benefits to full use of HART communication. Costs can be reduced by improving plant operations and increasing efficiency, and expensive process disruptions and unplanned shutdowns are avoided. Properly utilized, the intelligent capabilities of HART-smart devices can keep plants operating at maximum efficiency. A real-time HART integration with plant control, safety, and asset management systems unlocks the value of connected devices and extends the capability of systems in detecting any problems with the device or its connection to the process, or interference with accurate communication between the device and system.

The world's leading suppliers of process automation control systems and instrumentation all support HART communication. Most automation system suppliers offer direct HART-enabled I/O and PC-based software applications to leverage the intelligence in HART-smart field instruments or devices for continuous device condition monitoring, real-time diagnostics, and multivariable process information.

(1) HART communication principles

HART communication occurs between two HART-enabled devices, typically a field instrument or device and a control or monitoring system. An analog measurement signal is used to transmit digital information, and an additional signal is modulated to the measurement signal by using the frequency shift keying (FSK) technique. With the FSK, the two frequencies of the additional signal; 1200 and 2200 Hz, to represent the bit values 1 and 0, respectively. This makes it possible to transfer additional information without affecting the analog measurement signal. As indicated by Figure 12.8, HART provides two simultaneous communication channels: the 4 20 mA analog signals and a digital signal. The 4 20 mA signal communicates the primary measured value (in the case of a field instrument or device) using the 4 20 mA current loop, the fastest and most reliable industry standard. Additional

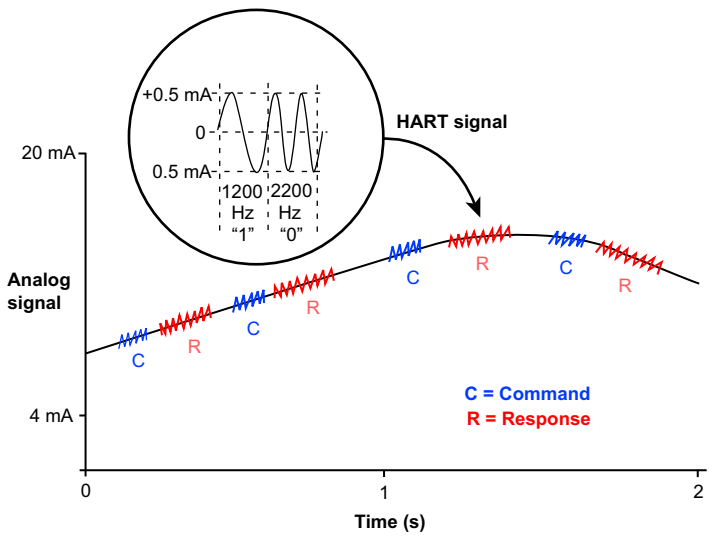


FIGURE 12.8

The HART signaling method including digital and analog.

device information is communicated using a digital signal that is superimposed on the analog signal containing information such as device status, diagnostics, additional measured or calculated values. Together, the two communication channels provide a complete field communication solution that is easy to use and configure, is low-cost and is very robust.

The HART signal path from the microprocessor in a sending device to the microprocessor in a receiving device is displayed in Figure 12.9. Amplifiers, filters, and the network between these two interfaces have been omitted for simplicity. At this level the diagram is the same, regardless of whether a master or slave is transmitting. Notice that, if the signal starts out as a current, the FSK is a voltage, but if it starts out a voltage it stays a voltage.

The transmitting device begins by turning on its carrier and loading the first byte to be transmitted into its interface circuits. It waits for the byte to be transmitted and then loads the next one. This is repeated until all the bytes of the message (these messages are always defined as commands that are of predefined format) are exhausted, then it waits for the last byte to be serialized and finally turns off its carrier. With minor exceptions, the transmitting device does not allow a gap to occur in the serial stream; the start and stop bits are used for synchronization and the parity bit is part of the HART error detection.

The serial character stream is applied to the modulator of the sending modem. The modulator operates such that a logic 1 applied to the input produces a 1200 Hz periodic signal at the modulator output. Logic 0 produces 2200 Hz. The type of modulation used is called continuous phase frequency shift keying (CPFSK). "Continuous phase" means that there is no discontinuity in the modulator output when the frequency changes. When the sender's interface output (modulator input) switches from logic 1 to logic 0, the frequency changes from 1200 to 2200 Hz with just a change in slope of the transmitted waveform. A moment's thought reveals that the phase does not change through this transition. Given the chosen shift frequencies and the bit rate, a transition can occur at any phase.

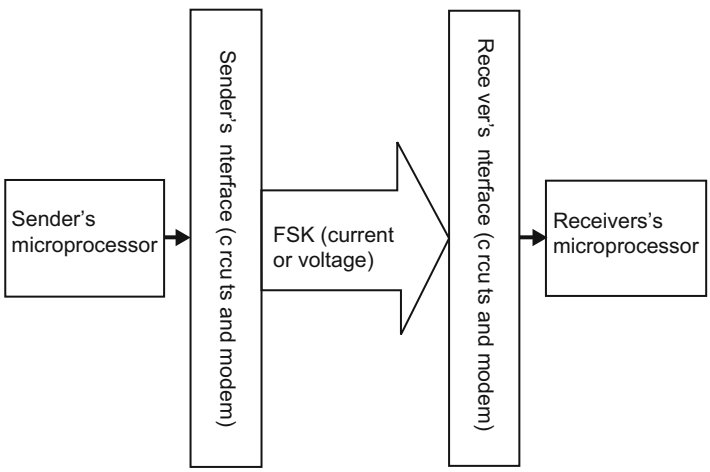


FIGURE 12.9

The HART signaling path including digital and analog.

At the receiving end, the demodulator section of a modem in its interface converts FSK back into a serial bit stream at 1200 bps. Each character is converted back into an 8-bit byte and parity is checked. The receiving microprocessor reads the incoming bytes from its interface and checks parity for each one until there are no more, or until parsing of the data stream indicates that this is the last byte of the message. The receiving microprocessor accepts the incoming message only if its amplitude is high enough to cause carrier detect to be asserted. In some cases, the receiving microprocessor will have to test an I/O line to make this determination. In others, the carrier detect signal gates the receive data so that nothing (no transitions) reaches the receiving interface unless carrier detect is asserted.

The HART protocol puts most of the responsibility (such as timing and arbitration) onto the masters. This eases the development of field instrument or device software and puts the complexity into the device that is more suited to deal with it. A master typically sends a command and then expects a reply. A slave waits for a command and then sends a reply. The command and associated reply are called a transaction. There are typically periods of silence (when no device is communicating) between transactions. A slave accesses the network as quickly as possible in response to a master request. Network access by masters requires arbitration, done by observing who sent the last transmission (a slave or the other master) and by using timers to delay their own transmissions. Thus, a master allows time for the other master to start a transmission. The timers constitute dead time when no device is communicating and therefore contribute to overhead in HART communication.

Each HART field instrument or device (in HART, a field instrument or device mostly plays the role of slave) must have a unique address, since each command contains the address of the desired field instrument or device and all field instruments or devices examine every command. The device which recognizes its own address sends back a response, which is then incorporated into the command message sent by a master and is echoed back in the reply by the slave. Addresses are either 4 bits or 38 bits and are called short and long or short frame and long frame addresses, respectively. A slave can also be addressed through its tag (an identifier assigned by the user).

Each command or reply is a message that starts with a preamble and is ended with a checksum. The preamble can vary in length, since different slaves can have different requirements; a master might need to maintain a table of these values. A master will use the longest possible preamble when talking to a slave for the first time. Once the master reads the slave's preamble, it first checks the length requirement (a stored HART parameter), then will subsequently use this new length when talking to that slave. The checksum at the end of the message is used for error control. It is the exclusive-OR of all of the preceding bytes, starting with the start delimiter. The checksum, along with the parity bit in each character, creates a message matrix having so-called vertical and longitudinal parity. If a message is in error, this usually necessitates a retry.

One more feature, available in some field instruments or devices, is burst mode. A field instrument or device that is burst-mode capable can repeatedly send a HART reply without receiving a repeated command. This is useful in getting the fastest possible updates (about 2 3 times per second) of process variables. If burst mode is to be used, there can be only one bursting field instrument or device on the network. A field instrument or device remembers its mode of operation during power down and returns to this mode on power up. Thus, a field instrument or device that has been parked will remain so through power down. Similarly, a field instrument in burst-mode will begin bursting again on power up.

(2) HART communication protocol

The HART protocol is widely recognized as the industry standard for digitally enhanced 4 20 mA field level communication in process control. It provides a uniquely compatible solution for field-level communication, as both 4 20 mA analog and digital signals are transmitted simultaneously on the same wiring.

HART communication uses a master-slave protocol, which means that a field device as slave speaks only when it is spoken to by a master. In every communication, a master device sends a command message first; the slave device processes it and then sends back a response message (Figure 12.10). Both the command and response messages include the HART data and must be formatted in accordance with the relevant HCF (HART Communication Foundation) specifications.

The HART protocol can be used in various modes for communicating information between devices, including analog plus digital signals, or digital only signals. Digital master-slave communication together with the 4 20 mA analog signals is the most common. This mode, depicted in Figure 12.10, allows digital information from the slave device to be updated twice per second in the master. The 4 20 mA analog signals are continuous and can still carry the primary variable for control. Please note that when a communication between the master and the slave is engaging, interrupt is definitely not allowed.

A burst is an optional communication mode (Figure 12.10) which allows a single slave device to continuously broadcast a standard HART response message. This mode frees the master from having to send repeated command requests to get updated process variable information. The same HART response message is continuously broadcast by the slave until the master instructs the slave to do otherwise. Data update rates of 3 4 per second are typical in burst-mode communication, but will vary with the chosen command. Please note that the burst mode should be used only in single slave device networks.

Two masters (primary and secondary) can communicate with slave devices in a HART network. Secondary masters, such as hand-held communicators, can be connected almost anywhere on the network and communicate without disturbing the primary master, typically a PLC, or computer-based central control or monitoring system. A typical installation with two masters is shown in

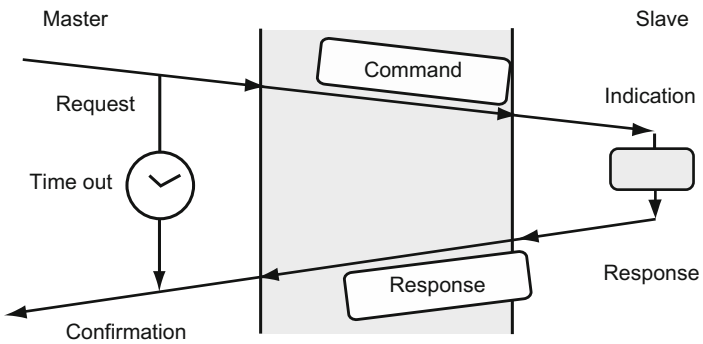


FIGURE 12.10

The HART master-slave protocol model.

OSI layer	Function	HART layer	
Application layer	Provides the user with network capable applications	Command oriented. Predefined data types and application procedures	
Presentation layer	Converts application data between network and local machine formats		
Session layer	Connection management service for applications		
Transport layer	Provides network independent, transport message transfer	Autosegmented transfer of large datasets, reliable stream transport, negotiated segment sizes.	
Network layer	End to end routing of packets. Resolving network addresses		Power optimized, redundant path, self healing wireless mesh network.
Data Link layer	Establishes data packet structure, framing, error detection, bus arbitration.	A binary, byte oriented, token passing, master slave protocol	Secure and reliable, time synched TDMA/CSMA, frequency agile with ARQ
Physical layer	Mechanical/electrical connection. Transmits raw bit stream	Simultaneous analog and digital signaling, normal 4 20 mA copper wiring.	2.4 Hz wireless, 802.15.4 based radies, 10 dBm Tx power
		Wired HART	Wireless HART

FIGURE 12.11

The layer structure of HART protocol in respect to the OSI 7-layer model.

Figure 12.12 and Figure 12.13. From an installation perspective, the same wiring that is used for conventional 4 20 mA analog instruments carries HART communication signals. Allowable run lengths will vary with the type of cable and the devices connected, but can in general be up to 3000 m for a single twisted pair cable with shield, and 1500 m for multiple twisted pair cables with a common shield. Unshielded cables can be used for short distances. Intrinsic safety barriers and isolators which pass the HART signals are readily available for use in hazardous areas.

HART protocol also has the capability to connect multiple field devices on the same pair of wires in a multi-drop network configuration, as shown in Figure 12.12(b). In multi-drop networks, communication is limited to master-slave digital only. The current through each slave device is fixed at the minimum value needed to power the device (typically 4 mA) and no longer has any meaning for the process.

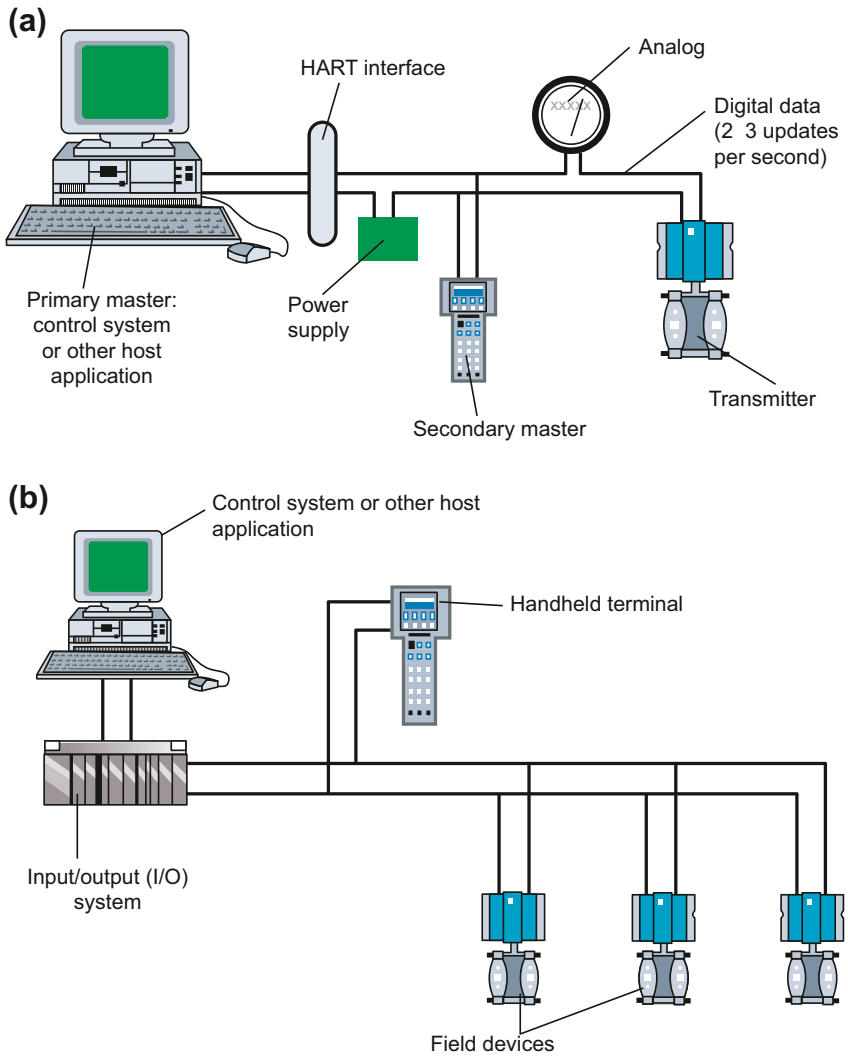


FIGURE 12.12

The architecture of HART networks: (a) the point-to-point HART network and (b) the multi-drop HART network. Note: Instrument power is provided by an internal or external power source that is not shown in this figure.

HART protocol utilizes the OSI 7-layer reference model. As is the case for most of the communication systems at field level, it implements only layers 1, 2, and 7 of the model. Layers 3 6 remain empty since their services are either not required or are provided by the application layer 7 (see Figure 12.11).

The application layer defines the commands, responses, data types, and status reporting supported by the protocol. In addition, there are certain conventions in HART (e.g., how to trim the

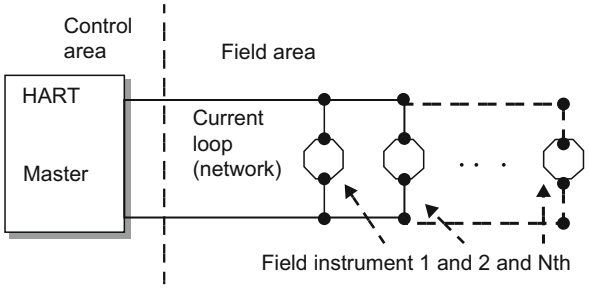


FIGURE 12.13

HART network with multi-drop field instruments.

network current) that are also considered part of the application layer. While the Command Summary, Common Tables, and Command Response Code Specifications all establish mandatory application layer practices (including data types, common definitions of data items, and procedures), the Universal Commands specify the minimum application layer content of all HART-compatible devices.

12.2.2 HART networks

HART communication comprises two folders: communication protocol and connection system. The previous subsection discussed the HART protocol. This subsection focuses on HART networks, and the next subsection will focus on HART devices.

There are two kinds of HART network available in industry: wired and wireless. Obviously, these two kinds of network use different connection media and different communication mechanisms, which therefore support different architectures and require different devices.

(1) Wired HART networks

There are two types of wired HART networks available their architecture is shown in Figure 12.12; the first (Figure 12.12(a)) is a point-to-point network, and the second (Figure 12.12(b)) is a multi-drop network. As shown in the figure, wired HART networks include the host controller and field devices that can be transmitters; between the host and the field devices, this system has an I/O system that can be the system interface with the HART network, and a hand-held terminal or hand-held communicator.

A network with a single field instrument or device that does both HART network functions and analog signaling is probably the most common type in use, and is called a point-to-point network. In some cases, it may have a HART field instrument or device but no permanent HART Master, for example, if the user intends to use mainly analog communication, and the field instrument or device parameters are set prior to system installation. A user might also set up this type of network and then later communicate with the field instrument or device which is using a hand-held communicator (also named HART secondary master). This is a device that clips onto device terminals (or other points in the network) for temporary HART communication with the field instrument or device.

A HART field instrument is sometimes configured so that it has no analog signal, only a HART function. Several such devices can be connected together (electrically in parallel) on the same network, as shown in Figure 12.13, in which case they are said to be multi-drop, and the master is able to talk to and configure each one in turn. When field instruments or devices are multi-drop there cannot be any analog signaling, and the term current loop ceases to have any meaning. Multi-drop field instruments powered from the network draw a small, fixed current (usually 4 mA) to maximize the possible number of devices. A field instrument or device that has been configured to draw a fixed analog current is said to be parked; accomplished by setting the short-form address of the field instrument or device to some number other than 0. A hand-held communicator might also be connected to the network, as displayed in Figure 12.13.

There are few restrictions on building wired HART networks. The topology may be loosely described as a bus, with drop attachments forming secondary buses as desired, as illustrated in Figure 12.14. The whole collection is considered a single network. Except for the intervening lengths of cable, all of the devices are electrically in parallel. The hand-held communicator (HHC) may also be connected virtually anywhere. As a practical matter, however, most of the cable is inaccessible and the HHC has to be connected at the field instrument or device, in junction boxes, or in controllers or marshalling panels. In intrinsically safe (IS) installations there is likely to be an IS barrier separating the control and field areas.

A field instrument may be added or removed, or wiring changes can be made while the network is live (powered). This may interrupt an on-going transaction. However, if the network is inadvertently short-circuited, this could reset all devices. The network will recover from the loss of a transaction by retrying a previous communication. If field instruments or devices are reset, they will eventually come back to the state they were in prior to the reset. No reprogramming of HART parameters is needed.

Digital signaling brings with it a variety of other possible devices and modes of operation. For example, some field instruments or devices are HART-only and have no analog signaling. Others draw no power from the network. In still other cases the network may not be powered (no direct current). There also exist other types of wired HART networks that differ from the conventional one described

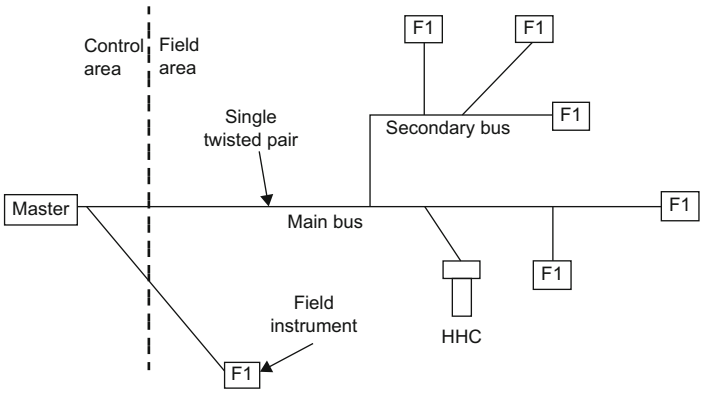


FIGURE 12.14

An HART network which shows a free arrangement of field devices.

here. These are covered in the technical reports and product manuals of the HART network manufacturers.

(2) Wireless HART networks

Wireless HART is the first open and interoperable wireless communication standard that has been designed to address the critical needs of the process industry for reliable, robust, and secure wireless communication.

A wireless HART network consists of field devices, at least one gateway, and a network manager, all wireless. These components are connected into a wireless mesh network supporting bidirectional communication. Figure 12.15 shows a typical wireless HART network architecture with only the principal devices plotted, which are as follows:

(1) Network manager

The network manager is the application software that manages the mesh network and network devices. It: (a) forms the mesh network; (b) allows new devices to connect to the network; (c) sets the communication schedule of the devices; (d) establishes the redundant data paths for all communications; and (e) monitors the network.

(2) Gateway devices

The gateway device connects the mesh network with a plant automation network, allowing data to flow between the two network devices. It provides access to the wireless HART devices for a system or other host application.

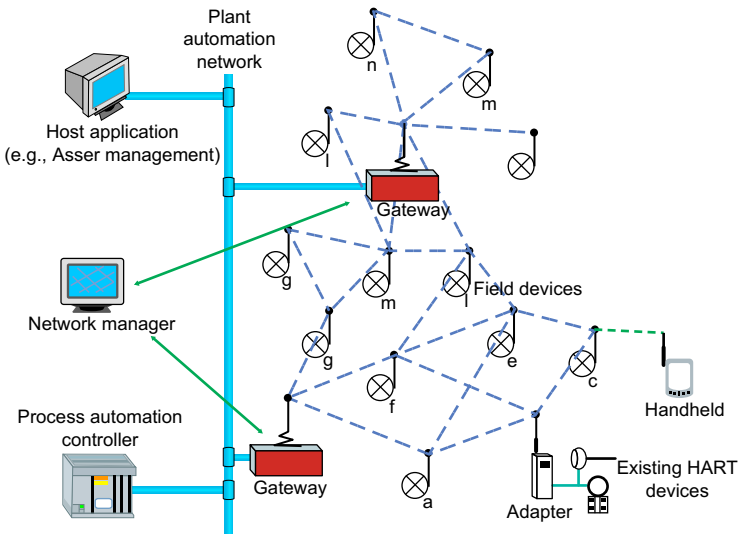


FIGURE 12.15

Typical wireless HART architecture.

(Courtesy of the HART Communication Foundation, in 2006.)

(3) Network devices

A network device is a node in the wireless mesh network. It can transmit and receive wireless HART data and perform the basic functions necessary to support network formation and maintenance. Network devices include field devices, router devices, gateway devices, and mesh hand-held devices.

(a) Field devices. The field device may be a process-connected instrument, a router, or hand-held device. The wireless HART network connects these devices together.
(b) Router device. A router device is used to improve network coverage (to extend a network), so it is capable of forwarding messages from other network devices.
(c) Process-connected instrument. It is typically a measuring or positioning device used for process monitoring and control; it is also capable of forwarding messages from other network devices.
(d) Adapter. An adapter is a device that allows a HART instrument without wireless capability to be connected to a wireless HART network.
(e) Hand-held support device. Hand-held devices are used in the commissioning, monitoring, and maintenance of network devices; they are portable and operated by plant personnel.

Wireless HART networks can be configured into a number of different topologies including the following:

1. Star network. Star networks have just one router that communicates with several end devices. This is one of the simplest network topologies, appropriate for small applications.
2. Mesh network. Mesh networks are formed by network devices that are all routers. They provide a robust network with redundant data paths, able to adapt to changing radio frequency environments.
3. Star mesh network. These are a combination of the above two.

12.2.3 HART devices

For building and maintaining a HART-enabled system, the technical kernel will be choosing the HART-compatible devices. Devices must be fully compliant with the HART protocol specification, and be tested and registered with the HART Communication Foundation (the HCF).

Devices which support the HART protocol are grouped into master (host) and slave (field) devices. Master devices include communicator or hand-held terminals as well as PC-based workplaces that stay in a control room. Slave devices, on the other hand, include sensors, transmitters, valves, and various actuators.

Both master (host) and slave (field) devices have an integrated FSK modem, whereas computers or workstations have a serial interface to connect the modem externally. HART communication is often used for such simple point-to-point connections. Nevertheless, in multi-drop connections, the number of accessible devices can be increased by using a multiplexer and some FSK buses. Figure 12.16 illustrates a multi-drop HART network which has a configuration of the FSK modem, HART multiplexer, and HART buses.

(1) HART communicator

The HART communicator is the type of communicator most widely used across the world in industrial control systems. They are portable, their weight has been evenly distributed for comfortable

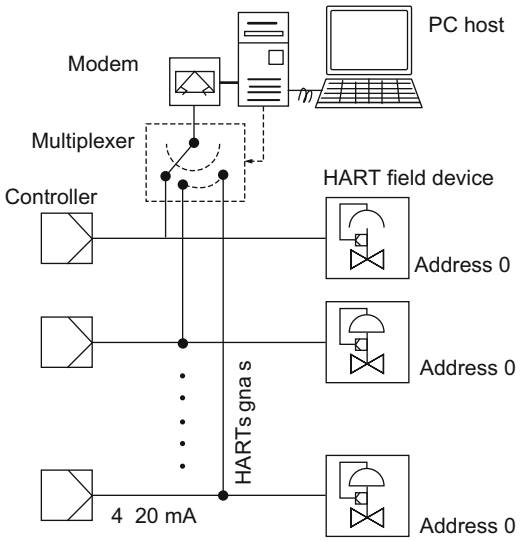


FIGURE 12.16

A HART connecting architecture for field devices, which includes the FSK modem and HART multiplexer and some HART buses.

one-handed operation in the field. The result is a universal, user upgradeable, intrinsically safe, rugged and reliable field communicator. In HART-enabled systems, such communicators are often defined by engineers as the secondary master (Figure 12.12(a)) or handheld terminal (Figure 12.12(b)).

Together with a memory and a microprocessor or some application-specific integrated circuits, the HART communicator provides a complete solution for configuring and monitoring all HART and fieldbus devices in an industrial control system.

It is composed of three main components plus accessories. These parts consist of the hand-held; the HART interface hardware, and the application software suite. The communicator runs on a robust, real-time operating system. This trio of hardware and software comprises a complete HART field communicator, which can be powerful, multifaceted, and portable all in one.

The hardware primarily includes the HART interface and associated pinch connectors. The interface is designed to join the multiple connectors located on the bottom of the hand-held device, allowing communication between it and the network. The pinch connectors easily connect to any HART network for instant communication. Most of the interface requires no batteries, running solely off the hand-held's internal power supply. Its compact size and low power consumption makes the interface very portable.

Its software suite includes some distinctive applications. The main application allows communication, monitoring, and configuration of HART-compatible devices, based upon manufacturer device description files (DDL) and thus allows access to all menus and parameters as designed by the manufacturer. The software allows the logging of device variable values over time. A wide range of variables can be logged automatically at a user-selectable sample time, or manually one by one. These logs can be saved and transferred to a PC for further analysis.

(2) FSK modem

There are often two kinds of modem required in the HART-enabled networks: the USB (universal serial bus) modem and the FSK (frequency shift keying) modem, all connected to the host PC (personal computer). The USB modem is an ordinary PC modem, without special adaptation for HART functions. However, the FSK modem should be particularly designed for HART functions. The following discussion will focus on the FSK modem.

The FSK modem is designed to provide HART communication capabilities for the implementation of frequency shift keying (FSK) techniques for data transfer. It is also required to conform to the HART network's physical layer. For this purpose, most FSK modems operate to the Bell 202 standard and are made into a chipset containing some integrated circuits. As shown in Figure 12.8, FSK is the frequency modulation of a carrier of digital capability. For simplex or half duplex operation, the FSK modem uses a single carrier in which communication can only occur in one direction at a time. For full duplex, the FSK modem uses multiple carriers for simultaneous communication in both directions.

The basic block diagram for the FSK modem chipset is depicted in Figure 12.17, which illustrates the mechanism of data modulation and demodulation. This chip is divided into three main parts: receive, transmit, and clock recovery. Both receive and transmit blocks are separate and data can be processed in each direction independently.

(1) Modulator for transmitting data

The data transmit section of this chipset performs modulation, in which the scrambler makes a nearly flat spectrum output signal, which is connected to a long digital FIR (finite impulse response) filter. This filter compensates for distortion in the transmission line and removes sharp edges a rising from high-low or low-high logic transitions. It makes the transmitted signal spectrum narrow to fit bandwidth and compensates the signal for the receiver's side. The transmit wave's shapes are stored in an EPROM (an electronic memory; refer to Chapter 5 of this textbook). In this way the transmitted waveform is synthesized not only from the present bit's state, but also from the four that preceded it and the four to come. Data burnt into EPROM represent filter response for each of 256 combinations.

(2) Demodulator for receiving data

At the receiving side, the audio signal going from the discriminator of the transceiver is passed through a low-pass filter to eliminate pertinent higher frequencies, and remove out-of-band spurious noise and

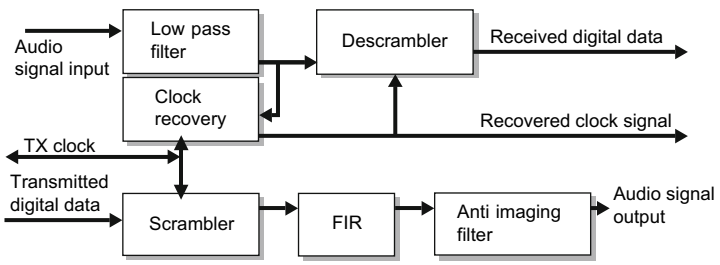


FIGURE 12.17

Block diagram of an FSK modem chipset.

residue. The signal is then limited and detected by sampling at the correct instant. At this point, the detected data, still randomized, are passed through a descrambler, where the original data are recovered and the result goes off to terminal. A descrambler, like a scrambler, simply provides the invert function of the scrambler and perforce requires some numbers of bits to synchronize.

(3) Clock recovery

The heart of the receiver is a digital phase-locked network (DPLL), which must extract a clock from the received audio stream. It is needed to time the receiver functions, including the all-important data detector. Each waveform has a phase shift of 360/256 degrees and is made up of 16 samples. The received audio signal is limited, and a zero-crossing detector circuit generates one cycle of 9600 Hz for each zero-crossing (a proto-clock). This is compared with a locally generated clock in a phase detector based on an up/down counter. The counter increments if one clock is early, decrements otherwise. This count then addresses an EPROM, mentioned above. In this way, the local clock slips rapidly into phase with that of the incoming data. The local clock signal is derived from the output of EPROM. Output data are converted to sine voltage with maximum amplitude.

(3) HART multiplexer

In HART-enabled industrial networks, the multiplexer acts as a gateway between the network management computer and the field devices. Many field devices are distributed over a wide area in industrial process systems, and must be monitored and adapted to changes in the processing environment. The HART multiplexer enables communication between an asset management computer or workstation and the field devices that support the HART protocol, in order to fit into the changes in the process system. All actions on the field device are parallel to the transmission of the 4 20 mA measurement signals and have no influence on the measurement value travelling through the system.

Each HART multiplexer, regardless of whether it is a slave or a master, provides a connection to a specified number of field instruments, allowing thousands of field units to communicate and exchange data with a computer or workstation. Working with a hand-held terminal (HART communicator) is also possible, since the HART protocol allows two masters (computer and hand-held terminal) to exist in one system. Systems can be easily expanded and the advantages of the HART communications can be exploited. Currently, a system can have a maximum of 31 HART multiplexer masters which are linked to the computer with an RS-485 interface. Each HART multiplexer master controls up to 15 HART multiplexer slaves.

(4) HART connecting buses

HART-enabled networks require several kinds of buses to connect devices and instruments. A brief description of two of these buses is given below.

(1) Bus for split-range operation

In industrial process systems, there are special applications which require that several (usually two) actuators receive the same control signal. A typical example is the split-range operation of control valves, in which one valve operates in the nominal current range from 4 to 12 mA, while another uses the current range from 12 to 20 mA. In this type of operation, control valves are connected in series in the network. When both valves have a HART interface, the host device must be able to determine

which valve it must communicate with. To achieve this, the HART protocol revision 6 (1999), and beyond will be extended by one more network variant. As is the case for multi-drop mode, each device is assigned to an address from 1 to 15. The analog 4 20 mA signals preserve its device-specific function, which is, for control valves, the selection of the required travel.

(2) FSK bus

The HART protocol can be extended by the FSK bus. Similar to a device bus, this can connect approximately 100 HART-compatible devices, and address these devices at the present technical level. This requires special assembly-type isolating amplifiers (e.g., TET 128). The limit on the number of participants is because each additional participant increases the signal noise lowering signal quality beyond the point that the telegram can be properly evaluated. The HART devices are connected to their analog current signal and the common FSK bus line with the isolating amplifier (Figure 12.18). From the FSK-bus viewpoint, these amplifiers act as impedance converters enabling devices with high load to be integrated into the network. To address these devices, a special, long form of addressing is used. During the configuration phase, the bus address and the tag number of each device are set with the point-to-point line. During operation, the devices operate with the long addresses. In this way, the system configuration can be read and checked during the start-up phase.

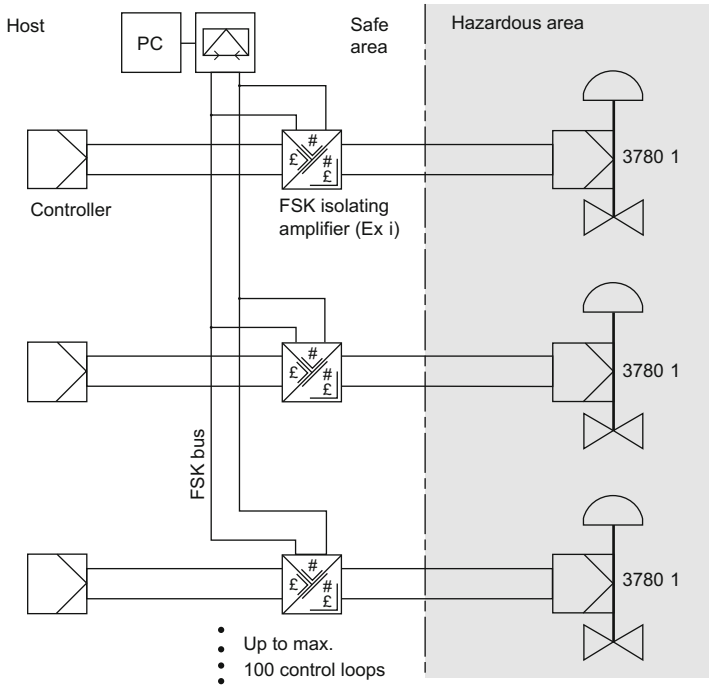


FIGURE 12.18

The connection architecture of a HART network with the FSK bus.

(Courtesy of Samson, Inc., in 2006.)

(5) HART system interface

HART communication can only function properly when all participants can interpret the HART signals correctly. To ensure this, not only must the transmission lines fulfil certain requirements, but also the devices in the current network must not impede data transmission. This is because the inputs and outputs of these devices are specified only for the 4 20 mA technology, and, as the input and output resistances change with the signal frequency, such devices are likely to short-circuit the higher-frequency HART signals (1200 2200 Hz).

Where a HART communication system is connected with other kinds of communication systems, gateways could be the best devices for protocol conversion. In most cases, when complex communications must be performed, fieldbus systems are preferred. Even though there is no complex protocol conversion, the HART-enabled system is capable of communicating over long distances, and the signals can be transmitted over telephone lines using HART/CCITT converters, in which the field devices directly connect to dedicated lines owned by the telephone company. However, as already mentioned, the HART-compatible field devices also require an appropriate communication interface that could be, for example, an integrated FSK modem or a HART multiplexer.

HART signals are imposed on the conventional analog current signal, whether devices are designed to use the four-wire or the two-wire technique. However, it is important to note that the maximum permissible load of a HART device is limited by the HART specification. Another limitation is due to the process controller, which must be able to provide the power for the connected two-wire device.

The higher the power consumption of a two-wire device is, the higher is its load. The additional functions of a HART communicating device increase its power consumption, and hence its load. When retrofitting HART devices into an existing installation, the process controller must be checked for its ability to provide the power required by the HART-compatible device; i.e., at least the load impedance of the HART device at 20 mA.

12.3 FIELDBUSES

12.3.1 Fieldbus systems

Fieldbuses were developed to replace the HART technology, discussed in section 12.2, in which centralized electronic instrumentation employs both 4 20 mA analog signals and a global digital communication standard. These two communication channels, analog and digital, simultaneously transmit through cables between hundreds of field devices and a central controller, thus enormous lengths of special cables may be needed.

Fieldbuses were expected to offer cost savings due to reduced wiring and terminations, short engineering and configuration times, fast loop-check-out and commissioning procedures and increased plant uptime resulting from advanced and fast diagnoses. Initially, there were several fieldbus standards approved by industrial organizations in various countries. To unify these standards, national organizations such as the Instrument Society of America, now the Instrumentation, Systems and Automation Society (ISA), the International Electro-technical Commission (IEC), Profibus (German national standard) and French national standard (FIP), formed the IEC/ISA SP50 Committee in 1990.

Fieldbus is a digital, bi-directional, multi-drop, serial communication network accommodating isolated control room and field devices. Each field device on the network has computing power

(intelligence) installed in it, making the device able to execute simple functions on its own, such as diagnostic, control, and maintenance functions, in additional to providing bi-directional communication capabilities.

Fieldbuses are wired in a completely different manner to HART. Communication with the latter can take place on several hierarchical levels: the management level, the control level, and the field level. A fieldbus system replaces the analog 4 20 mA current loops with a simple two-wire line running from the control station to the field devices, connecting all devices in parallel. Information is transmitted entirely digitally; control and process monitoring data as well as the commands and parameters required for start-up, device calibration and diagnosis.

The flexible fieldbus systems enable the connection of completely different field devices that can be controlled discontinuously as well as continuously operating sensors, actuators and valves. However, such a wide spectrum of applications is not always required. When only switching states need to be transmitted (such as simple sensors, solenoid valves, etc.), the relevant system components can be networked via an appropriately simplified bus system. For applications in hazardous areas, the open bus system AS-I (Actuator Sensor Interface) is a good solution, which can, if required, be the AS-I network integrated via special connection into more powerful fieldbus systems. An additional advantage of fieldbus technology is the considerable gain in functionality and safety. Apart from easy start-up and self-diagnosis, which is also true of smart HART devices, the fast fieldbus communication is also suitable for real-time-capable control systems, as both system status and error messages can be analyzed simultaneously.

The above list of advantages shows that the use of fieldbus networks changes the distribution of tasks between the central controllers and the field devices. As a consequence, the field devices operate more autonomously and are therefore equipped with microelectronic components.

(1) Fieldbus types and specifications

More than 100 different types of fieldbus are available, of which only a few have become established as standard. These work with a family of industrial computer network protocols for real-time distributed control, now standardized as IEC-61158. This was the final program of international fieldbus standards that was established in 2000. From Table 12.1 we learn that IEC-61158 is composed of six parts. Parts 1 and 2 are an introductory guide, a physical layer specification and service definition. Parts 3 and 4 are a data-link service definition and a data-link protocol specification, and finally, parts 5 and 6 are an application layer service definition and protocol specification.

In addition, there are eight types defined in the IEC-61158 standard. Type 1 through type 8 are Foundation Fieldbus H1, ControlNet, Profibus (Process Fieldbus), P-net, Foundation Fieldbus High-Speed Ethernet (HSE), World-FIP, and Interbus-S, respectively. This standard is going to have a significant effect on the development of the fieldbus.

The fieldbuses defined in the IEC-61158 standard are those most important for industrial applications.

(1) Foundation Fieldbus

The Foundation Fieldbus is the leading organization dedicated to a single, international, interoperable fieldbus standard. Foundation Fieldbus is a digital, serial, 2-way communications system that interconnects field devices such as sensors and actuators, with programmable controllers. At the field level in a plant control network, it routinely serves as a local-area network (LAN) for process control. It has

Table 12.1 An Introduction to the IEC-61158 Specifications

International Electro technical Commission (IEC: http://www.iec.ch): Industrial communication networks Fieldbus specifications: IEC 61158 Edition 1.0 (2007 12 to 2010 12)

Part Titles	Descriptions
Part 1: Overview and guidance for the IEC-61158 and IEC-61784 series	It presents an overview and guidance for the IEC-61158 series. It explains the structure and content of the IEC-61158 series; relates the structure of the IEC-61158 series to the ISO/IEC-7498 OSI Basic Reference Model; shows the logical structure of the IEC-61784 series; shows how to use parts of the IEC-61158 series in combination with IEC-61784 series; provides explanations of some aspects of the IEC-61158 series that are common to the parts of the IEC-61158 5 series. Its maintenance, resulting in the 2010 edition (IEC-61158 Edition 1.0), includes the following significant changes from the previous edition: deletion of the former Type 6 fieldbus, a placeholder for a Type 5 fieldbus data-link layer, and the Type 1 application layer for lack of market relevance; addition of new types of fieldbuses: types 11 to 20; generalization of the Type 1 radio, which is seldom used, to a more useful form; additional descriptions explaining the relation of IEC-61158 to the IEC-61784 family of companion profiles and the structure of these profile documents; division of parts 3 through 6 of the third edition into multiple parts numbered 3 1, 3 2, 3 19; 4 1, 4 2, 4 19; 5 2, 5 3, 5 20; 6 2, 6 3, 6 20; presentation of the service description concepts used in the many application layer service definitions: parts 5 2, 5 3, 5 20.
Part 2: Physical layer specification and service definition maintenance	It specifies the requirements for fieldbus component parts. It also specifies the media and network configuration requirements necessary to ensure agreed levels of data integrity before data-link layer error checking and interoperability between devices at the physical layer. The fieldbus physical layer conforms to layer 1 of the OSI 7-layer reference model as defined by ISO 7498 Specification with the exception that, for some types, frame delimiters are in the physical layer while for other types they are in the data-link layer.
Part 3: Data-link layer service definition:	It provides common elements for basic time-critical messaging communications between devices in an automation environment. The term “time-critical” is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life. The Part 3 of IEC-61158 Specification for data-link layer service definition includes these types: Types 1 (IEC-61158-3-1); Types 2 (IEC-61158-3-2); Types 3 (IEC-61158-3-3); Types 4 (IEC-61158-3-4); Types 7 (IEC-61158-3-7); Types 8 (IEC-61158-3-8); Types 11 (IEC-61158-3-11); Types 12 (IEC-61158-3-12); Types 13 (IEC-61158-3-13); Types 14 (IEC-61158-3-14); Types 16 (IEC-61158-3-16); Types 17 (IEC-61158-3-17); Types 18 (IEC-61158-3-18); Types 19 (IEC-61158-3-19).
Part 4: Data-link protocol specification	The data-link layer provides basic time-critical messaging communications between devices in an automation environment. Corresponding to each type for data-link layer service definition specified in Part 3 of IEC-61158 Specification, Part 4 of IEC-61158 Specification for data-link layer protocol specification includes these types: Types 1 (IEC-61158-4-1); Types 2 (IEC-61158-4-2); Types 3 (IEC-61158-4-3); Types 4 (IEC-61158-4-4); Types 7 (IEC-61158-4-7); Types 8 (IEC-61158-4-8); Types 11 (IEC-61158-4-11); Types 12 (IEC-61158-4-12); Types 13 (IEC-61158-4-13); Types 14 (IEC-61158-4-14); Types 16 (IEC-61158-4-16); Types 17 (IEC-61158-4-17); Types 18 (IEC-61158-4-18); Types 19 (IEC-61158-4-19).

(*Continued*)

Table 12.1 An Introduction to the IEC-61158 Specifications *Continued*

Part Titles	Descriptions
	(1) Type 1 provides the data-link service by making use of the services available from the physical layer. The relationship between the International Standards for fieldbus data-link service, fieldbus data-link protocol, fieldbus physical service and systems management is described in IEC/TR 61158-1. (2) Type 2 provides communication opportunities to all participating data-link entities, sequentially and in a cyclic synchronous manner. Foreground scheduled access is available for time-critical activities together with background unscheduled access for less critical activities. (3) Type 3 provides communication opportunities to a pre-selected 'master' subset of data-link entities in a cyclic asynchronous manner, sequentially to each of those data-link entities. Other data-link entities communicate only as permitted and delegated by those master data-link entities. (4) Type 4 provides a means of connecting devices through a partial mesh network, such that most failures of an interconnection between two devices can be circumvented. In common practice the devices are interconnected in a non-redundant hierarchical manner reflecting application needs. (5) Type 7 provides communication opportunities to all participating data-link entities in a synchronously starting cyclic manner, according to a pre-established schedule, and in a cyclic or acyclic asynchronous manner, as requested each cycle by each of those data-link entities. Thus this protocol can be characterized as one which provides cyclic and acyclic access asynchronously but with a synchronous restart of each cycle. (6) Type 8 provides a highly optimized means of interchanging fixed-length input/output data and variable-length segmented messages between a single master device and a set of slave devices interconnected in a loop (ring) topology. The exchange of input/output data is totally synchronous by configuration, and is unaffected by the messaging traffic. (7) Type 11 provides communication opportunities to all participating data-link entities in a synchronously starting cyclic manner, according to a pre-established schedule, and in a cyclic or acyclic asynchronous manner, as requested each cycle by each of those data-link entities. Thus this protocol can be characterized as one which provides cyclic and acyclic access asynchronously but with a synchronous restart of each cycle. (8) Type 12, Type 13, Type 14, Type 16, Type 18, and Type 19 provide communication opportunities to all participating data-link entities in a synchronously starting cyclic manner, and in a cyclic or acyclic asynchronous manner, as requested each cycle by each of those data-link entities. Thus this protocol can be characterized as one which provides cyclic and acyclic access asynchronously but with a synchronous restart of each cycle. (9) Type 17 provides communication opportunities to all participating data-link entities in a cyclic asynchronous manner, sequentially to each of those data-link entities, and in a synchronous manner, either cyclically or acyclically, according to a pre-established schedule. The specified protocol also provides means of changing the set of participating data-link entities and of modifying the set of scheduled communication opportunities. When the set of scheduled communication opportunities is null, the distribution of communication opportunities to the participating data-link entities is completely asynchronous.
Part 5: Application layer service definition	The Part 5 of IEC-61158 Specification for Application layer service definition includes these types: Types 2 (IEC-61158-5-2); Types 3 (IEC-61158-5-3); Types 4 (IEC-61158-5-4); Types 5 (IEC-61158-5-5); Types 7 (IEC-61158-5-7); Types 8 (IEC-61158-5-8); Types 9 (IEC-61158-5-9); Types 10 (IEC-61158-5-10); Types 11 (IEC-61158-5-11); Types 12

(IEC-61158-5-12); Types 13 (IEC-61158-5-13); Types 14 (IEC-61158-5-14); Types 15 (IEC-61158-5-15); Types 16 (IEC-61158-5-16); Types 17 (IEC-61158-5-17); Types 18 (IEC-61158-5-18); Types 19 (IEC-61158-5-19); Types 20 (IEC-61158-5-20).

(1) Type 2 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 2 fieldbus. The term time-critical is used to represent the presence of a time-window within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time-window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

(2) Type 3 is one of a series produced to facilitate the interconnection of automation system components. It is related to other standards in the set as defined by the three-layer fieldbus reference model described in IEC/TR 61158 1. This sub-part contains material specific to Type 3 fieldbus.

(3) Type 4 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 4 fieldbus.

(4) Type 5 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 5 fieldbus.

(5) Type 7 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment.

(6) Type 8 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 8 fieldbus.

(7) Type 9 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 9 fieldbus.

(8) Type10 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to type 10 fieldbus.

(*Continued*)

Table 12.1 An Introduction to the IEC-61158 Specifications *Continued*

Part Titles	Descriptions
	(9) Type 11 provides user programs with a means to access the Fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This part of IEC-61158 provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 11 fieldbus. (10) Type 12 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 12 fieldbus. (11) Type 13 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 13 fieldbus. (12) Type 14 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 14 fieldbus. (13) Type 15 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 15 fieldbus. (14) Type 16 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 16 fieldbus. (15) Type 17 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 17 fieldbus. (16) Type 18 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 18 fieldbus. (17) Type 19 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 19 fieldbus.

	(18) Type 20 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 20 fieldbus.
Part 6: Application layer protocol specification	In correspondence to each types for application layer service definition specified in the Part 5 of IEC-61158 Specification, the Part 6 of IEC-61158 Specification for Application layer protocol specification includes these types: Types 2 (IEC-61158-6-2); Types 3 (IEC-61158-6-3); Types 4 (IEC-61158-6-4); Types 5 (IEC-61158-6-5); Types 7 (IEC-61158-6-7); Types 8 (IEC-61158-6-8); Types 9 (IEC-61158-6-9); Types 10 (IEC-61158-6-10); Types 11 (IEC-61158-6-11); Types 12 (IEC-61158-6-12); Types 13 (IEC-61158-6-13); Types 14 (IEC-61158-6-14); Types 15 (IEC-61158-6-15); Types 16 (IEC-61158-6-16); Types 17 (IEC-61158-6-17); Types 18 (IEC-61158-6-18); Types 19 (IEC-61158-6-19); Types 20 (IEC-61158-6-20). (1) Type 2, Type 8, Type 9, Type 10, Type 11, Type 13, Type 14, Type 15, Type 16, Type 17, Type 18, Type 19, and Type 20 respectively provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 2 fieldbus. The term 'time-critical' is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time-window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life. (2) Type 3 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 3 fieldbus. (3) Type 4 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs' This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 4 fieldbus. (4) Type 5 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 5 fieldbus. (5) Type 7 provides user programs with a means to access the fieldbus communication environment. In this respect, the fieldbus application layer can be viewed as a window between corresponding application programs. This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 7 fieldbus.

a built-in capability to distribute control applications across the network, its communication models can be client/server, publisher/subscriber, and event notification, and it commonly exists in a three-layer model corresponding to the OSI reference model. The physical layer changes an electrical signal into a physical signal, the data-link layer controls messages transmitted on the fieldbus, and the application layer interfaces between the data-link layer and the flexible manufacturing system, using message transmission by standard message form.

There are two kinds of fieldbus protocol in Foundation Fieldbus: Foundation Fieldbus H1 and Foundation Fieldbus HSE (H2).

Foundation Fieldbus H1 is type 1 in IEC-61158, and is used in control applications such as the control of temperature, level, and flow. The speed of H1 is 31.25 kbps, 1 Mbps or 2.5 Mbps. It is also able to supply power for field devices directly and use the existing lines for 4 20 mA analog devices. Physical media are twisted-pair or fiber-optic. Network topology is star or bus. Data transfer size is 128 bytes. MAC (medium access control) is scheduling. Error checking is 16 bit cyclical redundancy checking (CRC). Maximum devices are 240/segment. The diagnostic method used is remote diagnostics.

Foundation Fieldbus HSE (H2) is a high-speed Ethernet protocol. The backbone network always runs at 100 Mbps. Although the H2 Fieldbus does not always use Ethernet protocol, it is generally used in the application of advanced process control, remote I/O and fast factory automation. It uses a 16 kHz alternating current signal to give support for supplying power using current mode. The signaling voltage of H2 is 5.5 to 9.0 Vp-p. Physical media are twisted-pair or fiber. Network topology is star. Data transfer size is varied and uses TCP/IP. Arbitration method is CSMA/CD. Error checking is CRC.

(2) Profibus

Profibus (or prifiBUS) was introduced in 1993. It utilizes a non-powered two-wire (RS-485) network which may have up to 126 nodes. It can transfer a maximum of 244 bytes data per node per cycle. Maximum communication (baud) rate is 12 Mbps with a maximum distance of 100 meters. The maximum distance is 1200 meters at 93.75 kbps without repeaters.

Profibus provides services at layers 1, 2 and 7 of the OSI model, although layer 2 also provides some functions of layers 3, 4 and 5. The physical layer (layer 1) can be RS-485 (2- or 4-wired cables) or fiber-optic.

Profibus-DP (Distributed Peripheral) is designed especially for communication between controllers and field devices at the field level. Profibus-DP can be used to replace parallel signal transmission with 24 V or 0 20 mA. Its transmission follows RS-485 or fiber-optic. Topology is bus. Physical media are twisted pair shielded copper cable. Transmission speed is 9.6 kbps at 1200 meters, 12 Mbps at 100 meters. Up to 127 nodes are connected with 32 devices within 1 segment.

Profibus-FMS (Flexible Manufacturing Systems) defines a large number of powerful communication services for both master/master (peer-to-peer) and master/slave communication models. The lower layer interface defines the representation of the FMS services on the data transmission protocol of layer 2. Topology is also bus.

Profibus-PA (Process Automation) permits field devices to be connected on one common bus line even in intrinsically safe areas. Profibus-PA permits data communication and power over the bus using 2-wire technology. Profibus-PA system is based on both Profibus-DP communication and IEC-61158-2 communication services. This fieldbus has the speed of 31.25 kbps.

(3) Interbus

Interbus is an open systems approach to a high-performance, ring-based, distributed device network for manufacturing and process control. The type 8 fieldbus defined in the IEC-61158 is the Interbus-S. The ring system topology actively connects devices via a closed-loop path with a transmission rate of 500 kbps. Due to the ring structure and because it has to carry the logic ground, Interbus-S requires a 5-wire cable between two devices. A distance of 400 meters between two devices is possible due to the RS-485 point-to-point transmission. The integrated repeater function in each device enables an overall extension of Interbus-S up to 13 km; maximum number of devices equals 512 nodes.

(4) World-FIP

World-FIP is the accepted fieldbus technology in a European Standard (EN-50170) which guarantees its stability and openness. World-FIP is a single communication technology for both time-critical data and unscheduled messages. A single communication technology serves each level of control architecture. Physical layer characteristics include topology standardized to IEC-1158-2; 31.25 kbps, 1 Mbps, or 2.5 Mbps data rates via twisted-pair copper cable or 5 Mbps via fiber-optic cable; 64 nodes per cable segment, with up to four segments through repeaters; and 1-kilometer (km) segment length, depending on data rate, cable, and number of nodes. World-FIP uses the producer/consumer model with a centralized bus scheduler.

(5) LonWorks

LonWorks operates over greater distances and is a practical peer-to-peer network, extensible to many thousands of points, though it can be comparatively slow and more complex. Two physical-layer signaling technologies, twisted-pair "free topology" and power line carrier, are typically included in each of the standards created around the LonWorks technology. The two-wire layer operates at 78 kbps using differential Manchester encoding, while the power line achieves either 5.4 or 3.6 kbps, depending on frequency.

(6) ControlNet

ControlNet is also an open network protocol, conceived as the ultimate high-level fieldbus network and designed to meet several high-performance automation and process control criteria. Of primary importance is the ability to communicate with 100% determinism, while achieving faster responses than traditional master/slave and poll-strobe networks.

At its physical layer, the network topology can be tree, bus, or star. ControlNet can operate with single or dual coaxial cable bus for cable redundancy. Maximum cable length without repeaters is 1 km and the maximum number of nodes on the bus is 99. Repeaters can be used to further extend the cable length. Fiber-optic cables can also be used.

At the data-link layer, ControlNet is a scheduled communication network designed for cyclic data exchange. The protocol operates in cycles, known as NUT (network update time). Each NUT includes two phases; the first dedicated to scheduled traffic, where all nodes with scheduled data are guaranteed a transmission opportunity, and the second dedicated to unscheduled traffic. There is no guarantee that every node will get an opportunity to transmit in every unscheduled phase. Both phases uses implicit token ring media access control. The end of each NUT is marked by the transmission of a moderator frame by the node. The maximum size of a scheduled or unscheduled ControlNet data frame is 510 bytes.

Table 12.2 A Comparison Between Some IEC-61158 Fieldbuses

Fieldbuses	Topology	Transfer Characteristics	Medium Access Control
Foundation Fieldbus H1	Bus	31.25 kbps, 1 Mbps and 2.5 Mbps	Scheduling (TDMA-based or other-based), multiple backup
Foundation Fieldbus H2	Star	100 Mbps	CSMA/CD
Profibus-DP/PA	Bus	DP: 12 Mbps PA: 31.25 kbps	Master/slave, and token
Profibus-FMS	Bus	9.6, 19.2, 93.75, 187.5, 500 kbit/s, 1.5, 3, 6 and 12 Mbit/s	Master/slave and peer-to-peer (master/master)
WorldFIP	Bus	31.25 kbps, 1, 2.5 Mbps with twisted-pair; 5 Mbps with fiber-optic	Producer/consumer with a centralized bus polling
Interbus-S	Ring	500 kbps, full duplex	Not required
LonWorks	Free topology	78 kbps	Peer-to-peer (master/master)
ControlNet	Bus, star and tree	5 Mbps	Concurrent TDMA

The fieldbuses defined in IEC-61158 have not only similar characteristics, but also have specific features. In Table 12.2, such fieldbuses are compared and analyzed with regard to topology, transfer characteristics and medium access control the most important aspects for performance. From this table, one can see that ControlNet is organized by topology of tree, bus, or star, so arrangement problems are thought to be under good control in ControlNet. It can also be seen that the Foundation Fieldbus HSE is very fast, at 100 Mbps, and it can be compared to the organization of the MAC (medium access controller) chipset. The Foundation Fieldbus H1 uses scheduling. The Foundation Fieldbus HSE uses CSMA/CD. Time division multiple access (TDMA) is also used for medium access control in fieldbus networks. Concurrent TDMA and synchronous TDMA are alike as they have the same speed. Interbus does not require medium access control.

(2) Fieldbus networks and protocols

The OSI reference model published by the International Standards Organization (ISO) is an established definition of network communications. It defines seven generic layers required by a communication standard capable of supporting vast networks. The first two layers in the OSI reference model, namely the physical and the data-link layers, incorporate the technologies to realize a reliable, relatively error-free, high-speed communication channel among the communicating devices. It provides support for all standard and medium-dependent functions for physical communication. The data-link layer manages the basic communication protocol as well as error control set up by higher layers.

In a fieldbus, communication takes place over fixed network routing, and transport layers are redundant. Moreover, in an industrial control environment, the network software entities or processes

are also generally invariant. In such a situation, requirements of the session and the presentation layers are also minimized, so the third, fourth, fifth and sixth layers of the OSI reference model have been omitted in fieldbus protocols, meaning most fieldbus networks utilize only three OSI model layers 1, 2 and 7. Below we discuss each of these three layers of the fieldbus in more detail.

(1) The physical layer

Fieldbus allows options for three types of communication media at this layer, namely, wire, fiber-optic and radio. The physical layer is divided into an upper and a lower section. The upper section ensures that the selected media interfaces in a consistent way with the data-link layer, regardless of the media used. The lower sections define the communications mechanism and media for example, for wire medium they describe signal amplitudes, communication rate, waveform, wire types, etc.

A field network can be implemented through the compartmentalization of the bus system in segments that can be connected over repeaters. Standard-transmission rates can be in the range of about 10 kbps to 10 Mbps. The topology of the single bus segment is a line structure (bus) with short drop cables. Transmission distances up to 12 km are possible with electrical configuration, and up to 23.8 km with optical configuration, with distances being dependent on transmission rate. With the help of repeaters, a tree structure can also be constructed. The maximum number of nodes per bus segment is 32. More lines can be connected under one another through performance enhancements (repeaters); it should be noted that each repeater counts as a node. In total a maximum of 128 nodes are connectable (over all bus segments).

(2) The data-link layer

The fieldbus data-link layer protocol is a hybrid, capable of supporting both scheduled and asynchronous transfers. Its maximum packet size is 255 bytes, and it defines three types of data-link layer entities: a link master, a basic device, and a bridge. Link master devices are capable of assuming the role of the bus master, also called the link active scheduler (LAS).

As mentioned before, the LAS controls communications traffic on the fieldbus. This is also called bus master function. The active LAS grants a right to transmit to each device on a fieldbus in a predefined manner. Devices other than LAS can communicate only when they have the right to transmit. There are two ways of granting a right to transmit; one is by polling, which grants this right to each device in sequence. Another is by time slot method, which grants the right at a fixed time interval. The LAS combines these two methods to meet the requirements of precise cyclic (scheduled) updates and acyclic (unscheduled) traffic.

Cyclic (scheduled) communications are used for regular, periodic transmission to a field device to publish (send message) information. All other devices on the fieldbus listen to the published information and receive the message if they are designated subscribers.

Acyclic communications send special commands such as retrieval of diagnostic information or acknowledgement of alarms to the designated fieldbus devices. Once a request for data transmission is initiated, the LAS will issue a special token to each device in turn, thereby allowing them access to the bus to transmit data or request data from another device, utilizing the bus up to an allocated time limit.

(3) The application layer

The application layer converts data and requests for services from the user into demands on the communication system in the layers below, and to provide the reverse service for received messages.

Thus, it abstracts the technical details of the network from the user, who can view the network devices to which communication is needed as if they are connected by virtual point-to-point communication channels.

There are two types of data transmission: byte-oriented and bit-oriented. Both protocols function equally well in automation applications. Byte-oriented protocols can be either asynchronous or synchronous, and bit-oriented protocols organize data packets into a series of bits, in which the packets are sent without any gaps in the message. The difference between bit- and byte-oriented transmission can be seen in the hardware used for each; only bit-oriented protocols have dedicated chips for communications. Both function equally well in automation applications.

There are three models used for communication in fieldbus networks: master/slave (client/server), peer-to-peer (master/master), and publisher/subscriber (scheduling). In the first, one master polls one or more slave devices using a request/response mechanism. Usually there is a single master, although some protocols allow more than one. Profibus can be designed with a primary master and a secondary master, and has an arbitration mechanism to allow multiple masters to share the fieldbus.

A different strategy is to use peer-to-peer networking which reduces the overhead on the master, compared with centralized or single-master solutions. In this case, each slave has time to use the bus, and can exchange data with other slaves without involving the master. A bus arbitration scheme is used to determine when a device can access the bus.

12.3.2 Foundation Fieldbus

In 1992, an international group, the ISP (Interoperable Systems Project), was founded. In 1994, the ISP and the FIP merged to form the Fieldbus Foundation with the intention of creating a single, international fieldbus standard for hazardous environments to use as the IEC-standardized fieldbus. The Fieldbus Foundation utilized some elements from the FIP and the Profibus-PA Fieldbus for the specification of their Foundation Fieldbus.

Foundation Fieldbus is an all-digital, two-way, multiple-drop communication system that brings the control algorithms into equipment and instrumentation. It has two communication protocols: H1 and HSE. The first, H1, transmits at 31.25 kbps and is used to connect the field devices. The second protocol, High Speed Ethernet (HSE), uses 10 or 100 Mbps Ethernet as the physical layer, and provides a high-speed backbone for the network.

H1 and HSE were specifically designed as complementary networks. H1 is optimized for traditional process control applications, whilst HSE, which employs low-cost Ethernet equipment, is designed for high-performance control applications and plant information integration. The combined H1/HSE Fieldbus solution allows for full integration of basic and advanced process control, and hybrid/batch/discrete control subsystems, with higher-level, supervisory applications. The combined H1/HSE Fieldbus solution provides optimized enterprise performance by removing unneeded I/O conversion equipment and controllers, sensor networks, and gateways. This flat, integrated architecture improves diagnostic methods and operator information, thus enhancing performance and reducing costs of controlled production systems.

In this system, the characteristic feature of distributed data transfer enables single field devices to execute automation tasks so that they are no longer "just" sensors or actuators, but contain additional functions. For the description of a device's function(s), and for the definition of uniform access to the data, the Foundation Fieldbus contains predefined function blocks to be discussed later on. When

Table 12.3 Some Function Blocks Defined in Foundation Fieldbus

Device types	Predefined Function Blocks
Sensors	Analog input or discrete input (digital input).
Control valves	Analog output or discrete output (digital output).
Controllers	Proportional/derivative (PD controller) or proportional/integral/derivative (PID controller).

implemented in a device, they provide information about the tasks that the device can perform. Typical functions provided by some types of device are shown in Table 12.3.

Foundation Fieldbus provides a shift of automation tasks from the control level down to the field level (Figure 3.1) resulting in the flexible, distributed processing of control tasks. Architected to a LAN network for all devices including field and control devices, Foundation Fieldbus supports digital encoding of data and many types of message. Unlike many traditional systems, which require a set of wires for each device, multiple Foundation Fieldbus devices can be connected to the same set of wires. It overcomes some of the disadvantages of proprietary networks by offering a standardized network. A simple fieldbus network setup is shown in Figure 12.19.

(1) Foundation Fieldbus H1

There are six conceptual parts to a fieldbus network: links, devices, blocks and parameters, linkages, loops, and schedules.

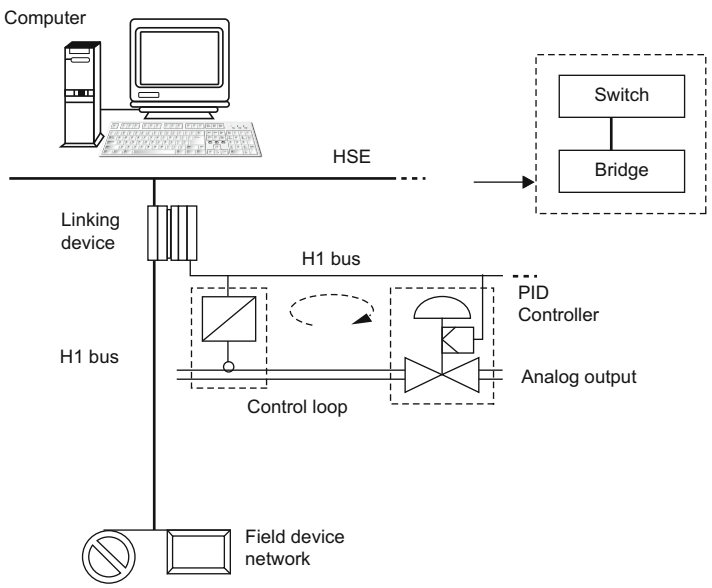


FIGURE 12.19

An industrial control system with Foundation Fieldbus.

(1) Links

A Foundation Fieldbus network is made up of devices connected by a serial bus. This serial bus is called a link (also known as a segment). A fieldbus network consists of one or more links, each configured with a unique link identifier.

The devices on the network can be field or host devices (PCs, workstation computers, or distributed control systems), configured with a physical device tag, an address, and a device identifier. The physical device tag must be unique within each fieldbus system, and the address must be unique within each link. The manufacturer assigns a unique identifier to the device. Figure 12.20 shows a link in a Foundation Fieldbus network.

(2) Devices

A key objective for Foundation Fieldbus is interoperability, so devices from a variety of manufacturers can be linked to take advantage of both the standard and the unique capabilities of every device. Instead of requiring that device manufacturers use only a given set of functions, Foundation Fieldbus uses device descriptions, which describe all the functions in a device. Using this, the host in a control system can obtain the information needed to create an interface that configures parameters, calibrates, performs diagnostics, and accomplishes other functions on the device.

There are three types of devices on a Foundation Fieldbus H1 network: link masters, basic devices, and H1 bridges.

(i) Link master. A link master device is capable of controlling traffic on a link by scheduling the communication on the network. Every fieldbus network needs at least one link-master-capable device either an interface board in a PC, a programmable controller, a distributed control system, or any other device, such as a valve or a pressure transducer. Link masters need not be separate devices; they can have I/O functionality (for example, you could buy temperature transmitters both with and without

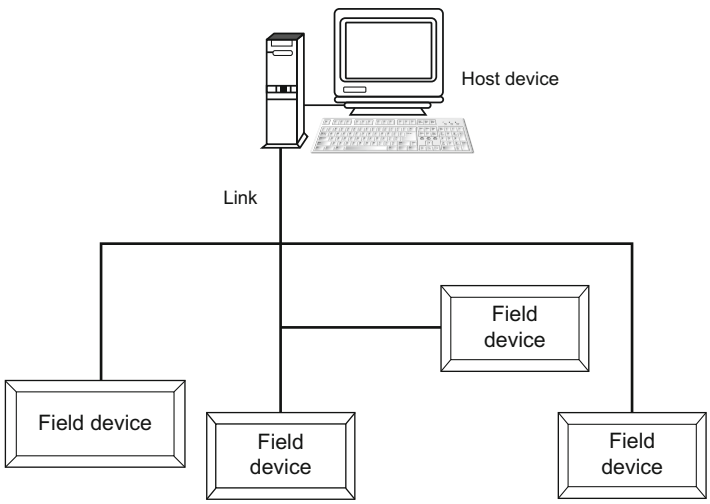


FIGURE 12.20

The link in a Foundation Fieldbus network.

link master capability). Fieldbus can operate independently of a computer system because of link masters on the bus, which have processing capability and are capable of controlling the bus. After you download a configuration to your device(s), your control loop can continue to operate even if the monitoring computer is disconnected.

All of the link masters receive the same information at the time of download, but only one link master will actively control the bus at a given time. The one currently in control is called the link active scheduler (LAS). If the current LAS fails, the next link master will take over transparently and begin controlling bus communications where the previous LAS left off, so no special configuration is required to implement redundancy. The LAS device follows the schedule downloaded during the configuration process. At the appropriate times, it sends commands to other devices, telling them when to broadcast data. It also publishes time information and grants permission to devices to allow them to broadcast unscheduled (acyclic) messages, such as alarms and events, maintenance and diagnostic information, program invocation, permissives and interlocks, display and trend information, and configuration.

(ii) Basic device. A basic device is not capable of scheduling communication. They cannot become the LAS.

(iii) H1 Bridge. Bridge devices connect links together into a spanning tree. They are always link master devices and they must be the LAS. An H1 bridge is a device connected to multiple H1 links whose data-link layer performs forwarding and republishing between and among the links. Note: be aware of the difference between a bridge and a gateway. While a bridge connects networks of different speeds and/or physical layers, a gateway connects networks that use different communication protocols. Figure 12.21 shows these three types of device.

(3) Blocks

Blocks can be thought of as processing units. They can have inputs, settings to adjust behavior, and an algorithm which they run to produce outputs, and they also know how to communicate with other blocks. The three types of blocks are the resource block, the transducer block, and the function block.

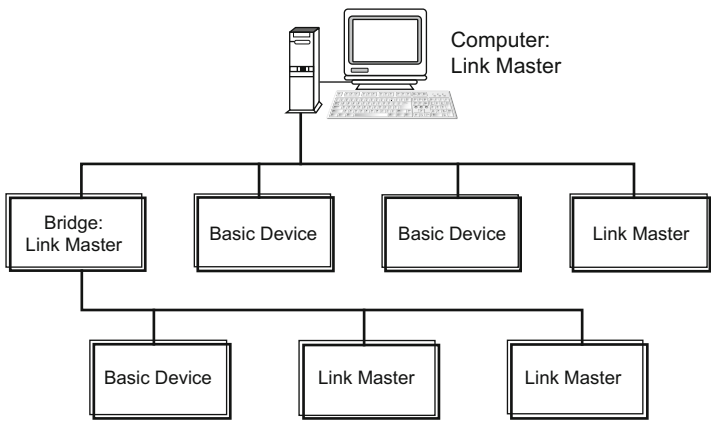


FIGURE 12.21

A classification of Foundation Fieldbus network devices.

(i) Resource block. A resource block specifies the general characteristics of the resource including the device type and revision, manufacturer identifier, serial number, and resource state. Each device has only one resource block, which also contains the state of all of the other blocks in the device. It must be in automatic mode for the device to execute. The resource block is a good place to start troubleshooting if the device is not behaving as desired, since it has diagnostic parameters to help determine the cause of problems.

(ii) Transducer blocks. Transducer blocks read from physical sensors into function blocks. Decoupling the function blocks from the hardware details of a given device, and so allowing generic indication of function block input and output.

The transducer block knows the details of I/O devices and how to read the sensor or change the actuator. It performs the digitizing, filtering, and scaling conversions needed to provide the sensor value to the function blocks, and/or makes the change in the output as dictated by the function block. Generally, there will be one transducer block per device channel, although in some devices, multiplexers allow multiple channels to be associated with one transducer block.

(iii) Function blocks. Function blocks provide the control and I/O behavior. Usually, a device has a set of functions it can perform represented as function blocks within the device. A function block can be thought of as a processing unit. They are used as building blocks in defining the monitoring and control application. The Foundation Fieldbus specification Function Block Application Process defines a standard set of function blocks, including 10 for basic control and 19 for advanced control. Table 12.4 shows the 10 function blocks for the most basic control and I/O functions.

Function block parameters, used for changing the behavior of a block, are classified as follows: input parameters receive data from another block; output parameters send data to another block; contained parameters do not receive or send data they are contained within the block.

(4) Linkages

The function blocks configured to control a process are linked, or connected by configuration objects inside the devices. These linkages allow data to be sent from one block to another. A linkage is different from a link, in that a link is a physical wire pair that connects devices on a fieldbus network, and a linkage is a logical connection that connects two function blocks.

Table 12.4 Ten Standard Function Blocks Defined in Foundation Fieldbus

Function Block Name	Symbol
Analog input	AI
Analog output	AO
Bias/gain	BG
Control selector	CS
Discrete input	DI
Discrete output	DO
Manual loader	ML
Propotional/derivative	PD
Propotional/integratral/derivative	PID
Ratio	RA

(5) Loops

A loop means a control loop which is a group of function blocks connected by linkages executing at a configured rate which sets the rate of execution, and also of data transfer. Figure 2.22 shows a control loop. It is possible to have multiple loops running at different rates on a link. Even if loops are running at different rates, they can send each other data through linkages. Figure 12.22 (a) and (b) show an example of multiple loops with linkage.

(6) Schedule

The schedule can be divided into two parts: a function block schedule that determines when a block executes, and a publishing schedule that determines when data parameters are published over the fieldbus. The function block schedule is downloaded to the particular device that contains it, and the publishing schedule is downloaded to a device or devices that have link master capability. As discussed earlier, the link master that is currently executing the publishing schedule, and thus controlling the process, is the link active scheduler.

(2) Foundation Fieldbus HSE

The Foundation Fieldbus HSE is based on standard Ethernet technology. The required components are therefore widely used and are available at low cost. When Fieldbus HSE runs at 100 Mbit/s, it cannot be equipped only with electrical lines, but needs fiber-optic cables as well.

Ethernet operates by using random (not deterministic) CSMA bus access. This method can only be applied to a limited number of automated applications because it requires real-time capability. The extremely high transmission rate enables the bus to respond sufficiently fast when the bus load is low and there are only a few devices.

If the bus load must be reduced due to the number of connected devices or if several HSE partial networks are to be combined to create a larger network, Ethernet switches must be used as shown by

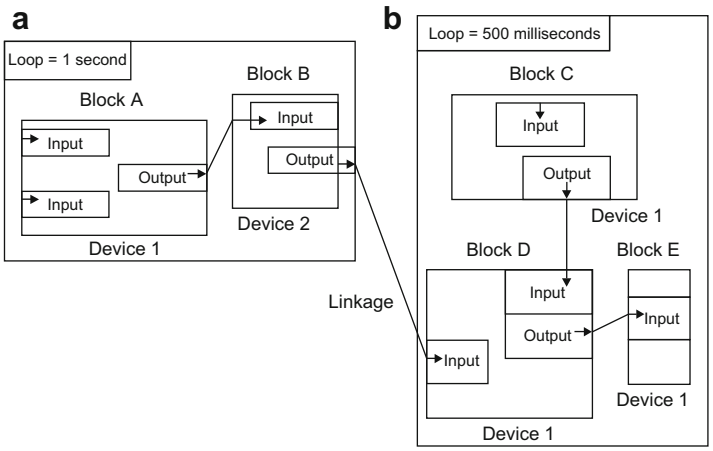


FIGURE 12.22

The control loops defined in Foundation Fieldbus H1 network: (a) shows a control loop; (b) shows multiple control loops. This figure shows multiple loops running at different rates, and having linkage between control loops.

the dashed square containing a switch and bridge in Figure 12.19. The switch reads the target address of the data packets that must be forwarded, and then passes the packets on to the associated partial network. This approach optimizes bus load and the resulting bus access time.

A communications network that consists of an H1 bus and an HSE network results in the topology illustrated in Figure 12.19 with the dashed square of switch and bridge. To connect the comparatively slow H1 segments to the HSE network, coupling components, so-called bridges, are required. Similar to HSE, the specification of this bus component has not yet been completed. A bridge is used to connect the individual H1 buses to the fast High-Speed Ethernet. The various data transfer rates and data telegrams must be adapted and converted, considering the direction of transmission. This way, powerful and widely branched networks can be installed in larger workshops and plants.

(3) Layered communications model

The Foundation Fieldbus specification is based on the layered communications model and consists of three major functional elements (Figure 12.23(a)): physical layer; communication stack; and user application.

The user application is made up of function blocks and the device description. It is directly based on the communication stack. Depending on which blocks are implemented in a device, users can access a variety of services. System management utilizes the services and functions of the user application and the application layer to execute its tasks (Figure 12.23(b) and (c)). It ensures the proper cooperation between the individual bus components, as well as synchronizing the measurement and control tasks of all field devices with regard to time.

The Foundation Fieldbus layered communications model is also based on the OSI reference model. As is the case for most fieldbus systems, and in accordance with the IEC specification, layers three to six are not used. The comparison in Figure 12.23 shows that the communication stack covers the tasks of layers two and seven, and that layer seven consists of the fieldbus access sublayer (FAS) and the fieldbus message specification (FMS).

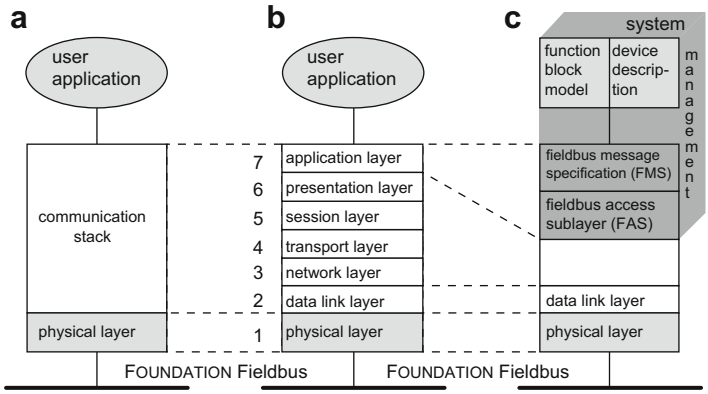


FIGURE 12.23

Three types of structure and description of the Foundation Fieldbus communication layers.

A device that sends data to other devices is called a publisher. A device that needs data from a publisher is called a subscriber. The LAS uses publisher/subscriber information to tell the publisher devices when to transmit their data over the bus. These scheduled data transfers are done in a deterministic manner, following the schedule developed in the configuration software and downloaded to the link masters.

12.3.3 Profibus Fieldbus

The history of Profibus (Process Fieldbus), or prifiBUS, goes back to an association venture project supported by the public authorities, which began in 1987 in Germany. Within the framework of this venture, 21 companies and institutions joined forces and created a strategic fieldbus project. Its goal was the realization and establishment of a bit-serial fieldbus, required for the standardization of the field device interface. For this purpose, the relevant member companies of the ZVEI (Central Association for the Electrical Industry) agreed to support a mutual technical concept for plant and process automation.

The first step of this effort worked out the specification of the complex communications protocol Profibus-FMS (Fieldbus Message Specification), which was tailored to fit demanding communication tasks. A further step in 1993 saw completion of the specification for the more simply configured and faster Profibus-DP (Decentralized Periphery) which is now available in three functionally scaleable versions; Profibus-DPV0, Profibus-DPV1 and Profibus-DPV2.

The third variant, Profibus-PA (Process Automation), was developed by the Profibus User Organization as a lower-speed, intrinsically safe counterpart to Profibus-DP for supplying the required hazard resistance of process automation. This variant is based on the services provided by the Profibus-DPV1, and is implemented as a partial system embedded in a higher-level Profibus-DP communication system.

International standardization of a fieldbus system is necessary for its acceptance, its establishment and its benefits. Profibus achieved national standardization in 1991 and 1993 in DIN-19245, Parts 1-3. However, these initial standards neglected the intrinsic safety and bus supply for Profibus, which greatly restricted its applications. Only when the international standard IEC-1158-2 was published in October 1994 did Profibus become a suitable transmission technique in industrial control systems. Then in 1996, it obtained Europe-wide standardization in EN-50170. Together with other fieldbus systems, Profibus has been standardized in IEC-61158 since 1999. In the course of these activities, the latest developments of Profibus were incorporated into this standard.

Now all three Profibus variants (Profibus-FMS, Profibus-DP, Profibus-PA) operate on one standardized bus access method. They are also able to use the same transmission technique (RS-485) and operate simultaneously on the same bus line. The transmission media used include either twisted-pair shielded cables for the Profibus-FMS or Profibus-DP, fiber-optics or radio waves. In the area of process automation, Profibus-PA connects process control stations and automated systems to field devices, thus replacing the analog 4 20 mA transmission technique. In addition to simple start-up and self-diagnostic functions, the fast fieldbus communication provides users with the option of realizing real-time-capable state control systems, as well as monitoring status and error messages parallel to the process. Coupled with the development of numerous application-oriented profiles and a rapidly growing number of devices, Profibus began its advance in factory automation and, since 1995, in process automation. Today, Profibus is a leading fieldbus in the world market.

(1) Profibus systems

The latest Profibus family consists of three compatible versions: Profibus-FMS (Fieldbus Message Specification), Profibus-DP (Decentralized Periphery), and Profibus-PA (Process Automation).

The initial version of Profibus was Profibus-FMS, designed to communicate between programmable controllers and computers at the control level (Figure 3.1) for sending complex information between them. Unfortunately, being the initial effort of Profibus designers, the Profibus-FMS technology was not as flexible as needed, and the protocol was not appropriate for less complex messages, or for communication on a wider, more complicated network. New types of Profibus-FMS have satisfied those needs so that it is still in use today, though the vast majority of users find newer solutions to be more appropriate.

The Profibus-DP was optimized for high speed and inexpensive hook-up. It has been designed and optimized particularly for communications between controllers or computers at the control level and decentralized field devices at the field level, so it communicates via cyclic data traffic exclusively. Each field device exchanges its input and output data with the control device within a given cycle time. In process engineering as well as in building process automation, operation and monitoring tasks require a visualization device in addition to the automation device. They require that device data be read or written during operation independently of the control cycle. Since the original Profibus-DP specifications did not provide any special services for these tasks, appropriate function extensions were defined in 1997. These extensions can be implemented optionally and are compatible with the existing Profibus-DP protocol and all earlier versions. The first extended Profibus-DP variant is referred to as Profibus-DPV0. In addition to the cyclic Profibus-DP communication services provided by Profibus-DPV0, Profibus-DP also offers acyclic services for alarm messages, diagnostics, parameterization and control of the field devices with other extended variants such as Profibus-DPV1 and Profibus-DPV2.

Profibus-PA is a protocol designed for process automation. In actuality, it is a type of Profibus-DP application profile which standardizes the process of transmitting measured data. Profibus-PA was designed specifically for use in hazardous environments. Unlike the automated applications in manufacturing engineering which require short cycle times of few milliseconds, other factors are of importance in process automation, such as intrinsically safe transmission techniques, field devices powered over the bus cable, reliable data transmission, and interoperability. To obtain these capabilities, Profibus-PA Fieldbus, along with its application profile in most environments, is able to support power over the bus.

These three variants of Profibus have been specified for different applications, as shown in Figure 12.24. Profibus-FMS is designed to communicate between programmable controllers and computers at the control level. Profibus-DP has been designed for communications between controllers or computers at the control level and decentralized field devices at the field level. Profibus-PA was designed specifically for use in hazardous environments at both control and field levels. The intrinsically safe Profibus-PA is usually part of a hierarchically structured network topology (Figure 12.24). It is connected to a Profibus-DP bus system on which also not intrinsically safe slaves and Profibus-PA bus masters operate via segment coupler.

Profibus is a smart fieldbus technology. Devices on the system connect to a central line, and once connected, these devices can communicate information in an efficient manner, and can go beyond automation messages. Profibus devices can also participate in self- and connection diagnosis. At the

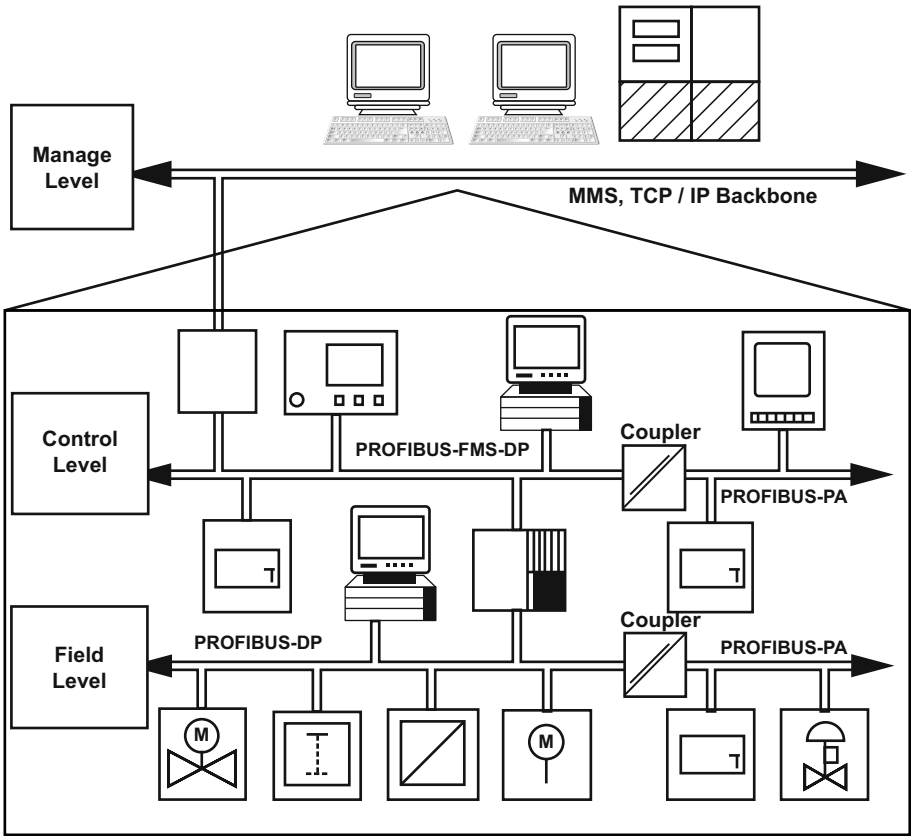


FIGURE 12.24

The application area and system architecture of Profibus Fieldbus including its three variants: Profibus-FMS, Profibus-DP, and Profibus PA.

most basic level, Profibus benefits from superior design of its OSI reference model layers and basic topology. Table 12.5 lists some of Profibus network parameters.

(2) Profibus transmission and communication

All variants of Profibus are based on the OSI reference model for communication networks. The layer structure of Profibus is shown in Figure 12.25. Due to field requirements, only OSI layers 1 and 2 are identical in these three variants of Profibus. The OSI layer 7 is only for the Profibus-FMS; the user interface of application programs is implemented for Profibus-DP and Profibus-PA.

(1) Physical layer

Profibus provides different versions of the physical layer as a transmission technology. All versions are based on international standards. There are two ways to realize the transmission technique for

Table 12.5 Some Profibus Network Parameters

Parameter Definition		Parameter Value
Type of network	Profibus-DP	Device bus
	Profibus-PA	Process control network
	Profibus-FMS	Control network (control level)
Physical media		Twisted pair, fiber-optic
Network topology		Bus, ring, star
Maximum devices	Profibus-DP	Maximum is of 126 stations on one bus (maximum of 244 bytes input and output data possible for each slave)
	Profibus-PA	Maximum is of 32 nodes/segment 4–6 per repeated segment depending on power requirements of devices and the type of barrier used
Maximum distance	Profibus-DP	93.75 kbps and less – 1200 meters 500 kbps – 400 meters 1.5 Mbps – 200 meters 12 Mbps – 100 meters
	Profibus-PA	1900 meters
	Max distance with repeater (maximum of 9 repeaters can be used)	9500 meters with repeaters
Communication methods	Profibus-DP	Peer-to-peer, multicast or cyclic master/slave (uses token passing sequence)
	Profibus-PA	Client/server, publisher/subscriber, event. Both scheduled and unscheduled communications
Device power supply	Profibus-DP	Devices are powered separately from communications bus
	Profibus-PA	Can be supplied from bus (typical)

Profibus-PA: either directly using the RS-485 standard, or in compliance with IEC-61158-2. When using the RS-485 interface, Profibus-FMS, Profibus-DP and Profibus-PA can be operated together on a common bus line. Intrinsically safe transmission in explosion-hazardous areas, however, requires the installation to be in accordance with IEC-61158-2. However, the masters of a Profibus-PA system, the control and operating stations, always operate on a Profibus-DP bus line in a safe area.

(i) Segment coupler. A bus coupler or segment coupler is installed between the Profibus-DP and the Profibus-PA segment. The coupler ensures electrical isolation between the safe and the intrinsically safe bus segment, powers of the Profibus-PA bus segment, adapts the transmission technique from RS-485 to IEC-61158-2, and converts between asynchronous and synchronous telegrams. In Figure 12.24 there are two couplers between a Profibus-DP and a Profibus-PA segment.

(ii) Network topology. All devices are connected in a bus structure (line). The maximum permissible line length depends on the transmission rate. Up to 32 stations (masters or slaves) can

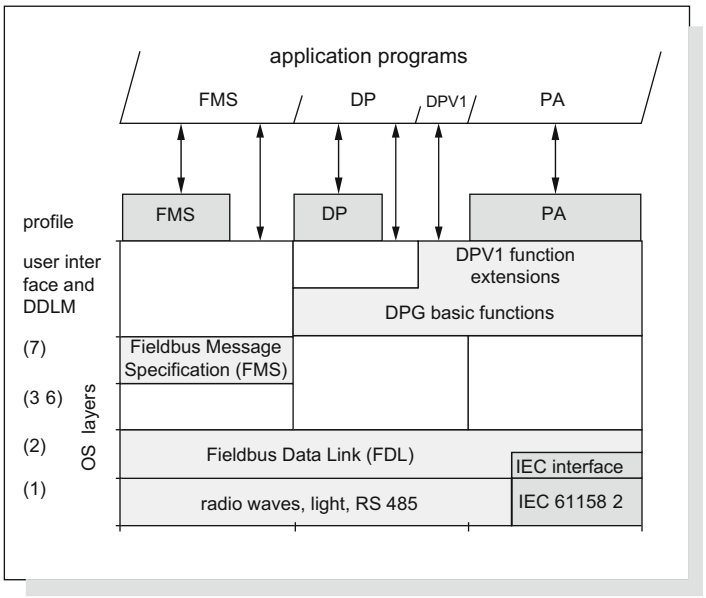


FIGURE 12.25

Layer structure of Profibus communication.

(Courtesy of Samson, Inc., in 2008.)

be connected in a single segment, the beginning and end of which are fitted with an active bus terminator. Both bus terminators have a permanent power supply to ensure error-free operation. The bus terminator is usually switched in the devices or in the connectors. If more than 32 stations are implemented, or there is a need to expand the network area, repeaters must be used to link the individual bus segments.

(iii) Cables and connectors. The properties of a fieldbus are also determined by the electrical specifications of the transmission cable. Although IEC 61158-2 does not specify a particular cable, the use of a reference cable is recommended (Type A, which has these parameters: twisted wire pair and shielded, 0.8 mm^2 wire diameter, 1900 meters cable length including stub lines). Only this type of cable enables data transmission over distances of up to 1900 meters. For an optimum electromagnetic compatibility, the bus lines must be shielded. This shield, as well as the metal cases of the field devices, must be grounded. Different cable types are available for connecting devices either to each other or to network elements (segment couplers, links and repeaters). Profibus cables are offered by a wide range of manufacturers.

(2) Data-link layer

The efficiency of the communication system is determined by the functions and services of the data-link layer, because it specifies significant tasks, such as the bus access control, the structure of data telegrams, basic communication services, etc. These tasks are performed by the fieldbus data link and fieldbus management.

A fieldbus data link manages bus access control (medium access control), telegram structure, data security, availability of data transmission services, send data with no acknowledge and send and request data with reply. Fieldbus management provides several management functions, for example: setting of operating parameters, report of events as well as the activation of service access points.

(i) Bus access and addressing. In Profibus communication, multiple-master systems are possible. The hybrid bus access control system operates on the token-passing method and uses the master/slave principle to communicate with the passive participants. Each master receives the token within a precisely defined time frame which allows it to have sole control over the communication network within that time frame.

(ii) Telegram structure. The Profibus-PA data telegrams of the IEC-61158-2 transmission are to a large extent identical with the Fieldbus Data Link telegrams of the asynchronous RS-485 transmission. Fieldbus Data Link defines the following: telegrams without data field (6 control bytes), telegrams with one data field of fixed length (8 data and 6 control bytes), telegrams with a variable data field (0 to 244 data bytes and 9 to 11 control bytes), brief acknowledgement (1 byte) and token telegram for bus access control (3 bytes).

(iii) Communication services. The data-link layer provides the application layer with send and request data with reply, and send data with no acknowledge communication services. For the former, the master issues a command or sends data to the slave and receives a reply within a defined time span. This reply either consists of an acknowledgement (brief acknowledgement) or is the requested data. In the send data with no acknowledge, the data are sent to a whole group of slaves. This permits event-controlled synchronization, where all slaves set their outputs simultaneously (synchronous mode) or update their input data simultaneously (freeze mode). A master-controlled bus assignment for slave replies is not possible in this case so that SDN telegrams remain unacknowledged. The access of the application to these basic forms of communication, as well as the various data-link services based on them is granted via so-called service access points which are used by the higher layers (user interfaces in some of Profibus networks) to perform all communication tasks of the respective application program.

(3) Profibus device management

The field devices in Profibus systems, as in other types of fieldbus systems, provide a wide range of information and also execute functions previously performed by programmable controllers and control systems. To do this, tools for commissioning, maintenance, engineering and parameterization of these devices require an exact and complete description of device data and functions, such as the type of application function, configuration parameters, range of values, units of measurement, default values, limit values, identification, etc. The same applies to the controller or control system, whose device-specific parameters and data formats must also be made known (integrated) to ensure error-free data exchange with the field devices.

The illustrations in Figure 12.25 show that the OSI layer 7 (application layer) is only used with Profibus-FMS; the OSI layers 3 to 6 are not used with these three Profibus variants. Figure 12.25 also shows that both Profibus-DP and Profibus-PA systems utilize a uniform user interface and also can be considered as standardized applications of the data-link layer (the OSI layer 2).

Therefore, the field device management of Profibus systems is a function layer that is defined as a “user interface” in many of related technical documents. It has several elements or descriptions, including the following.

(1) Device database files

Profibus has developed a number of methods and tools for this type of field device description which enable standardization of device management. The performance range of these tools is optimized to specific tasks, which has given rise to the term scaleable device integration. Therefore the tools are put together in one specification for Profibus with three volumes, which are: generic station description (GSD), field device tool (FDT), and electronic device descriptions (EDD). These methods and tools are routinely used in master stations, such as controllers and computers, at the control level for both Profibus-DP and Profibus-PA.

The interface description based on FDT, and the device description EDD, both aim at enabling the class-2 master to represent and operate the full scope of functions of all field devices. The most flexible way to operate is to load and update them when still in the project planning phase (see Figure 12.26).

(i) Generic station description (GSD). A GSD defines a device database file, provided by the device manufacturer and containing a description of the field devices either in Profibus-DP or in the Profibus-PA system. These files allow an open configuration tool to obtain device manufacturer and device identification number, transmission rate and bus parameters, number and format of the data for cyclic communication (for instance, cyclic positioner data), reference variable, controlled variable, final position feedback and status messages (fail-safe position, control loop fault, on-site operation, etc.). There are two ways to use these files. The first applies to compact devices whose block configuration is

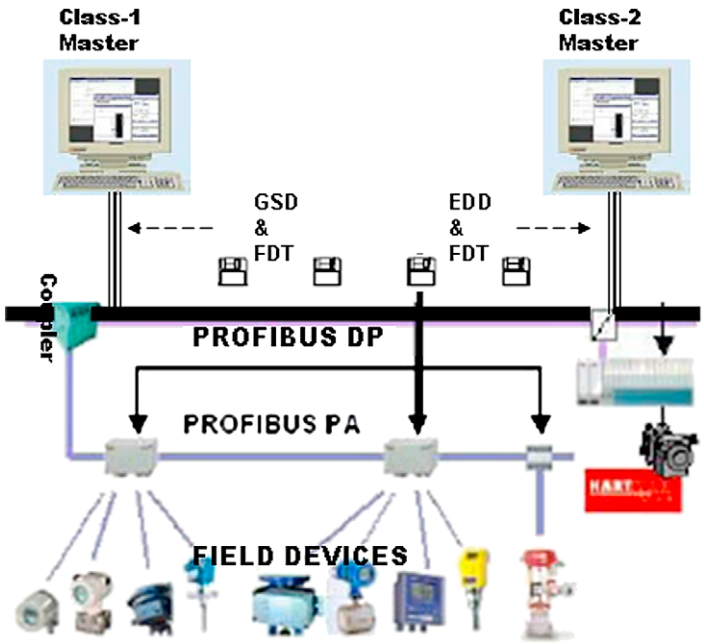


FIGURE 12.26

Loading the device database files as well as manufacturer-specific device and interface descriptions.

already known, and the second is for modular devices whose block configuration is not yet conclusively specified on delivery. In the latter case, the user must use the configuration tool to configure them in accordance with the actual module configuration.

(ii) Field device tool (FDT). The existing description languages for configuration and parameterization of field devices require an "auxiliary tool" that allows device manufacturers to provide users with expanded and specific characteristics of their field devices in standardized form over standardized interfaces. The definition of an auxiliary tool provides a universal interface with the ability to implement suitable software components on all automation systems fitted with it. Such an interface has been specified and designated a field device tool.

(iii) Electronic device descriptions (EDD). The GSD is inadequate for describing application-related parameters and functions of a field device (for example configuration parameters, ranges of values, units of measurement, default values, etc.). However, process automation required a more powerful description language, which has been developed in the form of the universally applicable EDD language. This language provides the tools to describe the functionality of field devices, and also contains support mechanisms to integrate existing profile descriptions in the device description, allow references to existing objects so that only supplements require description, allow access to standard dictionaries, and allow assignment of the device description to a device. Using this language tool, device manufacturers can create the relevant file for their devices which, like the GSD file, supplies device information to the engineering tool and then subsequently to the control system.

(2) Device profiles

Profibus achieves standardization of all device functions and parameters as well as the access to these data by using so-called device profiles, which determine how to implement communication objects, variables and parameters for the different types of field devices.

(i) Classification of device parameters. The field device parameters and data which can be accessed through communication can be divided into three groups: process parameters, operating parameters, and manufacturer-specific parameters. As an example, these three groups of parameters for a control valve are partially given in Figure 12.27.

(ii) Function block model. With device profiles, Profibus operates on the basis of a function block model. This model groups the different device parameters into several functional blocks which ensure uniform and systematic access to all parameters. Due to its object-oriented assignment of device parameters and device functions, the function block model simplifies planning and operation of distributed automation systems. Additionally, this model ensures compatibility with the international fieldbus standard so that a conversion to an international fieldbus protocol would not require modification of the application software.

The function block model assigns dynamic process values and the operating and standard parameters of a field device to different blocks (Figure 12.28). The function block describes the device function during operation (cyclic data exchange of analog input/output, alarm limit values, etc.). The physical block encompasses all parameters and functions required to identify the hardware and software (revision numbers, limit values, etc.). The transducer block contains the parameters which describe coupling signals to the process and are required to pre-process the data in the field device (process temperature and pressure, characteristic curves, sensor type, etc.). The operating mode, including start-up, operation, maintenance or diagnosis, determines which

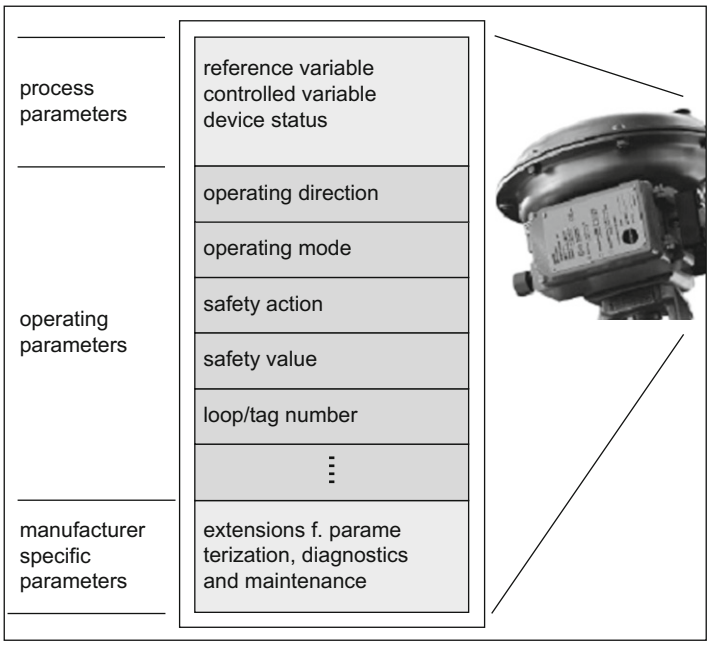


FIGURE 12.27

Classification of the Profibus device parameters for control valves.

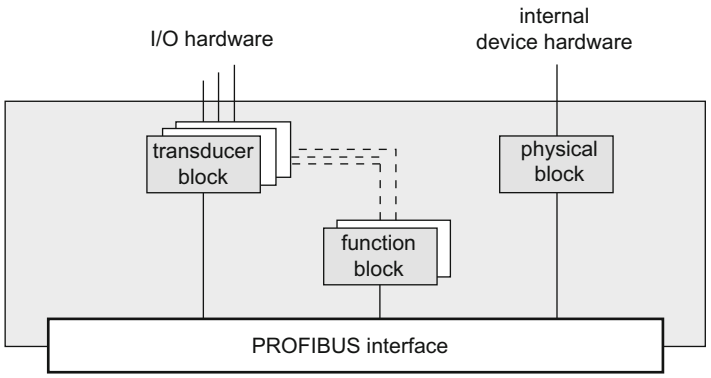


FIGURE 12.28

The function block model of Profibus device profiles.

parameters and blocks must be used. For instance, during operation, function block parameters are used almost exclusively, while during maintenance and start-up, transducer and physical block parameters are used. For diagnosis, information is required from all three blocks. During operation, a transducer block can be assigned to each function block. The process and system data saved in the

transducer blocks can be used by the field device to pre-process its own data and, thus, to provide the master with extended process information. The more extensive the transducer blocks defined by the device profile, the more varied is the process information provided by the respective field device.

Problems

1. Field communication technology in industrial systems is an important part of industrial control technology. Please write a short essay reviewing the development and evolution of field communication technology in industrial systems since 2000.
2. Please explain why: (1) AS Interface is an open system; (2) AS Interface has two cables on which both data and power are simultaneously transmitted; (3) AS Interface is a serial transmission network; (4) AS interface is able to handle both digital and analog data; (5) AS Interface offers an asynchronous coupling between a master device and all its slaves.
3. Please give a complete definition of the AS Interface.
4. All versions of the AS Interface specification are managed by AS International, a member funded organization located in Germany. Please make your comments on three versions of AS Interface specification: the original version (1994, version 2.04); the enhanced version (1998, version 2.11); and the modified version (2007, version 3.0).
5. An AS Interface network of either type 1 or type 2 requires several components falling into the following general categories: (1) gateways (also called masters); (2) power supplies and repeaters; (3) I/O modules (also called slaves); (4) accessories including cables. Please write a functional decomposition for each of these components.
6. There are three aspects affecting how AS Interface works in real time applications: connectivity, cycle time, and availability. Please analyze these three aspects and explain how they can make the AS Interface work in real time applications.
7. The master slave principle utilizes the polling technique: the master polls all AS Interface slaves one after the other and waits for responses. Suppose you have a field network of three sensors and three actuators which communicates with a programmable controller via the AS Interface. Based on Figure 12.4, please write a user program either in C++ or C to execute polling in this field network.
8. Based on Figure 12.5, please give the data image for the field network given in problem 7 above.
9. Please comment on the operation phase given in Figure 12.6 and the data flow given in Figure 12.7 for AS Interface.
10. All the system limits for the AS Interface in this chapter change dynamically with technical developments. Please refer to relevant specifications and manuals to give the current values of the system limits for AS Interface.
11. In terms of the HART signal path from the microprocessor in a sending device to the microprocessor in a receiving device in Figure 12.9, please give an explanation for the assertion that if the signal starts out as a current, the FSK is a voltage; but if the signal starts out as a voltage, the FSK stays a voltage.
12. In continuous phase frequency shift keying (CPFSK), there is no discontinuity in the modulator output when the frequency changes. When the sender's interface output (modulator input) switches from logic 1 to logic 0, the frequency changes from 1200 to 2200 Hz with just a change in slope of the transmitted waveform. Please explain: (1) why the phase does not change through this transition in CPFSK; (2) why, given the chosen shift frequencies and the bit rate, a transition can occur at any phase.
13. With the modulator for the FSK modem given in Figure 12.17, the data transmit part in this chipset performs modulation. The output of the scrambler is connected to a long digital FIR (finite impulse response) filter. The FIR filter makes the transmitted signal spectrum narrow to fit bandwidth and compensates the signal for the receiver's side. The transmit wave's shapes are stored in an EPROM memory. In this way the transmitted waveform is synthesized not only from the present bit's state, but also the four that preceded it and four to come. Data burnt into EPROM represents filter responses for each of 256 combinations. Please explain how it can calculate 256 combinations.

14. Please list the differences in general between fieldbus technologies and HART technologies used for field control systems; and compare their advantages and disadvantages as far as you can.
15. Please explain why most fieldbus networks just require the protocol layer 1, 2, and 7 of the OSI reference model for their data communications.
16. Please explain why the maximum length of cable or wire depends upon the data transmission rate in all fieldbus systems.
17. Please plot two schematic diagrams: one is for the wiring structure of conventional point to point communication systems; the other is for the wiring structure of general fieldbus communication systems.
18. Plot the protocol layer architecture of fieldbus networks in respect to that in the OSI reference model.
19. Which part of the IEC 61158 specification gives the standards for Foundation Fieldbus H1? The updated versions of the IEC 61158 specification include but are not limited to 16 different protocol sets called "types"; which of these 16 types is for the Foundation Fieldbus HSE?
20. The Foundation Fieldbus solves pending communication tasks by using two bus systems, the slow H1 bus and the fast HSE bus (Figure 12.19). Please explain, as you understand it, why the H1 bus is located at low level connecting with field devices; and the HSE bus is located at high level with host devices in Foundation Fieldbus networks.
21. What is the link in fieldbus networks? What is a linking device in fieldbus networks? What is a link master in fieldbus networks?
22. There are six conceptual parts to a Foundation Fieldbus network: links, devices, blocks and parameters, linkages, loops, and schedules. Please explain, as you understand it, the functional relations between any two of them.
23. By searching through the industrial equipment market, find as far as possible equipment that is currently used in the Foundation Fieldbus systems.
24. In Foundation Fieldbus communication models (Figure 12.23), the user application is made up of function blocks and device descriptions. It is directly based on the communication stack. Depending on which blocks are implemented in a device, users can access a variety of services. Please plot a simple diagram to express the correspondences of the services with the blocks.
25. Based on Figure 12.19 for Foundation Fieldbus and Figure 12.24 for Profibus Fieldbus, please list as far as you can the differences between the systems or networks, communication models and device management, etc. of these two types of fieldbus.
26. Foundation and Profibus Fieldbuses look similar at first on the physical layer; the main differences between them are signaling methods although both Fieldbuses follow the IEC 61158 2 signaling specifications. Please explain the technical details of the signaling methods of these two types of fieldbuses.
27. In those computer networks using the OSI 7 layer model, the data communications basically controlled by the data link layer (layer 2) and are executed by both the network layer (layer 3) and the transport layer (layer 4). The mechanisms in these computer networks are the routing table established by the network topology and address configurations, and the specific routing devices such as routers. However, most fieldbus networks do not use the OSI layers 3, 4, 5 and 6. The data communications in fieldbus networks are controlled and executed by the data link layer, which are mostly depend on the carrier sense multiple access with collision detection (CSMA/CD) mechanism and modes such as master/slave, publisher/subscriber, peer to peer, etc. Please explain how each of these modes (master/slave, publisher/subscriber, and peer to peer) works with the CSMA/CD mechanism in fieldbus networks.
28. Please investigate these topics for World FIP Fieldbus: variants, system or network architectures, communication models, and field device managements.
29. Please investigate these topics for Interbus Fieldbus: variants, system or network architectures, communication models, and field device managements.
30. Please investigate these topics for LonWorks Fieldbus: variants, system or network architectures, communication models, and field device managements.
31. Please investigate these topics for ControlNet Fieldbus: variants, system or network architectures, communication models, and field device managements.

Further Reading

Samson (http://www.samson.de). Communication in the field. http://www.samson.de/pdf en/l450en.pdf. Accessed: June 2009.

Wikipedia (http://en.wikipedia.org). AS Interface. http://en.wikipedia.org/wiki/AS Interface. Accessed: June 2009.

Pepperl+Fuchs (http://www.am.pepperl fuchs.com). AS Interface. http://www.am.pepperl fuchs.com/products/productfamily.jsp?division=FA&productfamily id=1402. Accessed: June 2009.

IDEC (http://www.idec.com). AS I Solution Kit: asi tutorial.pdf. January 2003.

Anybus (http://www.anybus.com). AS Interface. http://www.anybus.com/technologies/asi.shtml. Accessed: June 2009.

AS INTERFACE (http://www.as interface.net). AS Interface System. http://www.as interface.net/System/. Accessed: May 2009.

AS INTERFACE (http://www.as interface.net). AS Interface Products. http://www.as interface.net/Products/. Accessed: May 2009.

Alan R. Dewey in the Emerson, Inc. 2005. HART, Fieldbus work together in integrated environment. http://www.emersonprocess.com/home/library/articles/protocol/protocol0507 teamwork.pdf. Accessed: October 2008.

Analog Services, Inc. (http://www.analogservices.com). 2006. HART book. http://www.analogservices.com/about part0.htm. Accessed: October 2008.

David Belohrad, Miroslav Kasal. 1999. FSK modem with GALs. http://www.isibrno.cz/~belohrad/radioelektronika99 fskmodem.pdf. Accessed: October 2008.

Honey Well (http://hpsweb.honeywell.com). 2008. Wireless HART. http://hpsweb.honeywell.com/Cultures/en US/Products/wireless/SecondGenerationWireless/default.htm. Accessed: October 2008.

H. Kirrmann in ABB Research Center of Switzerland. 2006. The HART protocol. AI 411 HART.ppt. Accessed: October 2008.

N. P. Mahalik (Ed.). Fieldbus Technology: Industrial Network Standards for Real Time Distributed Control. Springer, Hamburg. 2003.

Bela G. Liptak (Ed.). Instrument Engineers' Handbook, 4th edition, Volume 2: Process Control and Optimiza tion. CRC Press. 2005.

Richard Zurawski (Ed.). The Industrial Communication Technology Handbook (The Industrial Information Technology Series). CRC Pres. 2005.

IEC (http://webstore.iec.ch). Reference IEC 61158. http://www.iec.ch/cgi bin/procgi.pl/www/iecwww.p?wwwlang=E&wwwprog=sea22.p&search=iecnumber&header=IEC&pubno=61158. Accessed: July 2009.

Jonas Berge. Introduction to Fieldbuses for Process Control. ISA Press. 2003.

Sang Geon Park. 2002. Fieldbus in IEC61158 Standard. Proceedings of the 15th CISL Winter Workshop (Kushu, Japan). pp. 200 205.

Wikipedia (http://en.wikipedia.org). Fieldbus. http://en.wikipedia.org/wiki/Fieldbus. Accessed: July 2009.

Samson (http://www.samson.de). Foundation Fieldbus. Technical Information Part 4. L454EN. 2005.

Samson (http://www.samson.de). Profibus PA. Technical Information Part 4. L453EN. 2005.

National Instruments (http://www.ni.com). Fieldbus: Foundation Fieldbus Overwiew. 2006.

PROFIBUS Nutzerorganisation e.V. PNO (http://www.profibus.com.de). PROFIBUS: System Description. Version October 2008.

Edward Red. Profibus Fieldbus. http://research.et.byu.edu/eaal/html/ProfibusFieldbus/body profibus fieldbus.html. Accessed: August 2009.

TopWorx (http://www.topworx.com). Profibus Overview. http://www.topworx.com/networx/pb.html. Accessed: August 2009.

CHAPTER

Human–machine interfaces 13

The term "user interface" refers to the methods and devices that are used to accommodate interaction between machines and the human who use them. The user interface of a mechanical system, a vehicle, a ship, a robot, or an industrial process is often referred to as the human machine interface. In any industrial control system, the human machine interface can be used to deliver information from machine to people, allowing them to control, monitor, record, and diagnose the machine system through devices such as image, keyboard, mouse, screen, video, radio, software, etc. Although many techniques and methods are used in industry, the human machine interface always accomplishes two fundamental tasks: communicating information from the machine to the user, and communicating information from the user to the machine.

Two types of user interface are currently the most common. Graphical user interfaces (GUI) accept input via input devices and provide an articulated graphical display on the output devices. Web user interfaces (WUI) accept input and provide output by generating web pages which are transmitted via the Internet and viewed by the user via a web browser. These types of user interfaces have become essential components in modern industrial systems, for example, human robot communication interfaces, SCADA human machine interfaces, road vehicle driver screens, aircraft and aerospace shuttle control panels, medical instrument monitors, etc.

13.1 HUMAN–MACHINE INTERACTIONS

Automated equipment has penetrated virtually every area of our life and our work environments. Human machine interaction is already playing a vital role across the entire production process, from planning individual links in the production chain right through to designing the finished product. Innovative technology is made for humans, used by and monitored by humans. The products therefore should be reliable in operation, safe, accepted by personnel, and, last but not least, cost-effective. This interplay between technology and user, known as human machine interaction, is hence at the very heart of industrial automation, automated control, and industrial production.

13.1.1 Models for human–machine interactions

Modelling the human machine interaction is done to depict how human and machine interact with each other in a system. It illustrates a typical information flow (or process context) between the "human" and "machine" components of a system. Figure 13.1 shows the components involved in each side of the human machine interaction. The environment side has three components: display, CPU, and I/O device components. The human side has another three components: sensory, cognitive, and musculoskeletal components.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10013-0

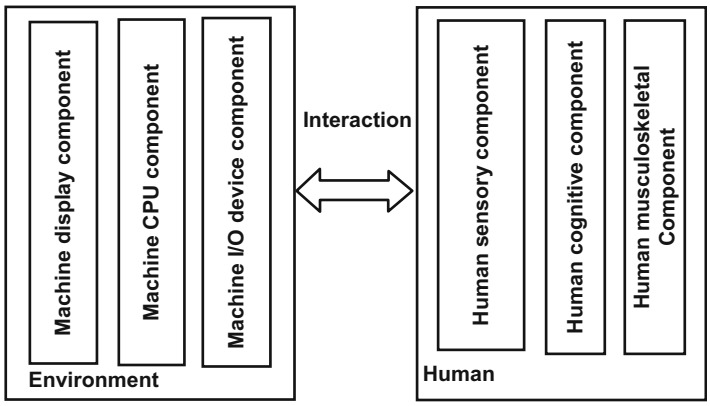


FIGURE 13.1

The components involved in human machine interactions.

At this point, we need to consider the process of human thinking (cognitive process) in the interaction. Figure 13.2 shows a model in which the operator's cognitive behavior and action mode differ in accordance with their perception of the information presented at the interface. As a result, he or she has three modes of action: skill-based, rule-based, and knowledge-based.

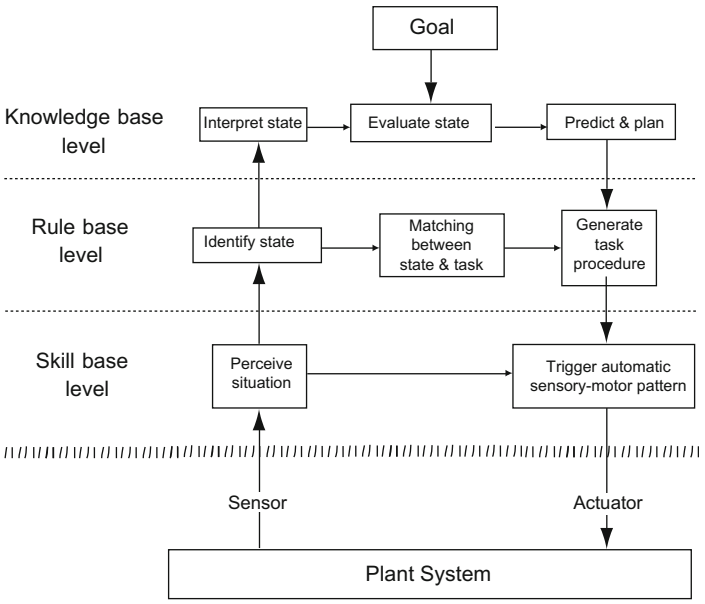


FIGURE 13.2

J. Rasmussen's operator model for human machine interactions in industrial control and automation.

(J. Rasmussen, Information Processing and Human machine Interaction, An Approach to Cognitive Engineeing. North Holland. 1986.)

In the skill-based mode, the operator acts unconsciously as smoothly as an automated control system. However, in the rule-based mode, the operator acts consciously and rapidly makes associative actions by remembering the relationship between the perceived state of situation and the corresponding action that is required. The relationship between the environmental situation, the state and the action is a mental model that the operator represents internally as a scheme. The skill-based and rule-based modes are automatic, thereby requiring minimal mental effort. Well-trained operators can perform actions in both models.

In contrast, mental models used in the knowledge-based mode differ from the associative relationship. Here, operators recognize the state of the plant by using their knowledge of the semiotic relationship of the plant's configuration: that is, of its goal and function, means and ends, parts and whole. They also recognize the logical relationship between factors such as the cause consequence relations between the input and output variables, as well as the qualitative relations between variables based on physical equations. By using these difficult, abstract models, operators can interpret the sensor information, find the root cause of any failure, predict future trends, and then decide what to do. This type of action is intense, laborious, and time-consuming, especially for a poorly trained novice. However, even well-trained operators can use this knowledge-based mode when they encounter unfamiliar situations.

(1) Definitions and constructs

In modern control systems, a model is a common architecture for grouping several machine configurations under one label. The set of models in a control system corresponds to a set of unique machine behaviors. The operator interacts with the machine by switching among models manually, or monitoring the automatic switching triggered by the machine. However, our understanding of models and their potential contribution to confusion and error is still far from complete. For example, there is widespread disagreement among user interface designers and researchers about what models are, and also how they affect users. This lack of clarity impedes our ability to develop methods for representing and evaluating human interaction with control systems. This limitation is magnified in high-risk systems such as automated cockpits, for which there is an urgent need to develop methods that will allow designers to identify the potential for error early in the design phase. The errors arising from modeling the human machine interaction are thus an important issue that cannot be ignored.

The constructs of the human machine interaction model discussed below are measurable aspects of human interaction with machines. As such, they can be used to form the foundation of a systematic and quantitative analysis.

(a) Model behaviors

One of the first treatments of models came in the early 1960s, from cybernetics; the comparative study of human control systems and complex machines. The first treatments set out the construct of a machine with different behaviors. In the following two paragraphs, we define a model as a machine configuration that corresponds to a unique behavior.

The following is a simplified description of the constructs of machine behaviors. A given machine may have several components (e.g., X1, X2, X3), each having a finite set of states. On Startup, for example, the machine initializes itself so that the active state of component X1 is "a," X2 is "f," and X3 is "k" (see Table 13.1). The vector of states (a, f, k) thus defines the machine's configuration on Startup. Once a built-in test is performed, the machine can move from Startup to Ready. The transition to Ready

Table 13.1 Machine Configurations with Different Behaviors

	X1	X2	X3
Startup	a	f	k
Ready	b	g	l
Engaged	c	h	m

can be defined such that for component X1, state "a" undergoes a transition and becomes state "b"; for X2 state "f" transitions to "g", and for X3, "k" changes to "l". The new configuration (b, g, l) defines the Ready model of the machine. Now, there might be a third set of transitions, for example, to Engaged (c, h, m), and so on.

The set of configurations labelled Startup, Ready and Engaged, if embedded in the same physical unit, corresponds to a machine with three different ways of behaving. A real machine whose behavior can be so represented is defined as a machine with input. The input causes the machine to change its behavior. One source of input to the machine is manual; the user selects the model, for example, by turning a switch, and the corresponding transitions take place. But there can be another type of input if some other machine selects the model, the input is "automatic". More precisely, the output of the other machine becomes the input to our machine. For example, a separate machine performs a built-in test and outputs a signal that causes the machine in Table 13.1 to transition from Startup to Ready, automatically.

(b) Model error and ambiguity

Model errors fall into a class of errors that involve forming and carrying out an intention (or a goal). That is, when a situation is falsely classified, the resulting action may be one appropriate for a perceived or expected situation, but is inappropriate for the actual situation. The type of situations leading to model error are not well known. For instance, consider a word processor, particularly when used in situations in which the user's input has different interpretations, depending on the model. For example, in one word processing application, the keystroke "d" can be interpreted as either the literal text "d" or as the command "delete". The interpretation depends on the word processor's currently active model: either Text or Command.

Model error can be linked to model ambiguity by interjecting the notion of user expectations. In this view, "model ambiguity" will result in model error only when the user has a false expectation about the result of his or her actions. There are two types of ambiguity: one that leads to model error, and one that does not. An example of model ambiguity that does lead to model error is timesharing operating systems in which keystrokes are buffered until a Return or Enter key is pressed. However, when the buffer is full, all subsequent keystrokes are ignored. This leads to two possible outcomes; either all or only a portion of the keystrokes will be processed. The two outcomes depend on the state of the buffer, which is either not full or full. Since this is unknown to the user, false expectations may occur. The user's action, hitting the Return key and seeing only part of what was keyed on the screen, is therefore a "model error" because the buffer has already filled up, but this was unknown to the user.

An example of model ambiguity that does not lead to model error is a common end-of-line algorithm for editing a word file which determines the number of words in a line. An ambiguity is

introduced because the criteria for including the last word on the current line, or wrapping to the next line, are unknown to the user. Nevertheless, as long as the algorithm works reasonably well, the user will not complain because he or she has not formed any expectation about which word will stay on the line or be placed on the next, and either outcome is usually acceptable. Therefore, model error will not occur even though model ambiguity does indeed exist.

(c) User factors: task, knowledge, and ability

One important element that constrains user expectations is the task at hand. If discerning between two or more different machine configurations is not part of the user's task, model error will not occur. Consider, for example, the radiator fan of your car. Do you know what configuration (OFF, ON) it is in? The answer, of course, is no. There is no such indication in most modern cars. The fan mechanism changes its mode automatically depending on the output of the water temperature sensor. Model ambiguity exists because at any point in time, the fan mechanism can change its model or stay in the current model.

The configuration of the fan is completely unknown to the driver. But does such model ambiguity lead to model error? The answer is obvious; not at all, because monitoring the fan configuration is not part of the driver's task. Therefore, the user's task is an important determinant of which machine configurations must be tracked and which need not be tracked.

The second element is user knowledge about the machine's behaviors. By this, we mean that the user constructs some mental model of the machine's response map. This mental model allows the user to track the machine's configuration, and, most importantly, to anticipate the next one. Specifically, our user must be able to predict what the new configuration will be following a manually or an automatically triggered event.

The problem of anticipating the next configuration of the machine reliably becomes difficult when the number of transitions between configurations is large. Another factor in user knowledge is the number of conditions that must be evaluated as TRUE before a transition from one model to another takes place. For example, the automated flight control systems of modern aircraft can execute a fully automatic (hands-off) landing. Several conditions (two engaged autopilots, two navigation receivers tuned to the correct frequency, identical course set, and others) must be TRUE before the aircraft will execute automatic landing. Therefore, in order to reliably anticipate the next model configuration of the machine, the user must have a complete and accurate model of the machine's behavior, including its configurations, transitions, and associated conditions. This model, however, does not have to be complete in the sense that it describes every configuration and transition of the machine. Instead, as discussed earlier, the details of the user's model must be merely sufficient for their task, which is a much weaker requirement.

The third element in the assessment of expectations is the user's ability to sense the conditions that trigger a transition. Specifically, the user must be able to first sense the events (e.g., a flight director is engaged; aircraft is more than 400 feet above the ground) and then evaluate whether or not the transition to a model (say, a vertical navigation) will take place. These events are usually made known to the user through an interface, but there are several control systems in which the interface does not depict the necessary input events. Such interfaces are said to be incorrect.

In large and complex control systems, the user may have to integrate information from several displays in order to evaluate whether the transition will take place or not. For example, one of the conditions for a fully automated landing in a two-engine jetliner is that two separate electrical sources

must be online, each one supplying its respective autopilot. This information is external to the automatic flight control system, in the sense that it involves another system of the aircraft. The user's job of integrating events, some of which are located in different displays, is not trivial. One important requirement for an efficient design is for the interface to integrate these events and provide the user with a succinct cue.

In summary, we have discussed three elements that help to determine whether a given model ambiguity will or will not lead to false expectations. First is the relationship between model ambiguity and the user's task. If distinguishing between models (e.g., radiator fan is ON or OFF) is not part of the user's task, no meaningful errors will occur. Second, in a case where model ambiguity is relevant to the user's task, we assess the user's knowledge. If the user has an inaccurate and or incomplete model of the machine's response map, he or she will not be able to anticipate the next configuration and model confusion will occur. Third, we evaluate the user's ability to sense input events that trigger transitions. The interface must provide the user with all the necessary input events. If it does not, no accurate and complete model will help; the user may know what to look for but will never find it. As a result, confusion and model error will occur.

(2) Classifications and types

The constructs described above are measurable aspects of human interaction with machines. As such, they can be used to form the foundation of a systematic and quantitative analysis. But before such an analysis can be undertaken, some form of representation, such as classification and types of models, is needed.

A classification of the human machine interaction models is proposed here to encompass three types of models in automated control systems: (1) interface models that specify the behavior of the interface, (2) functional models that specify the behavior of the various functions of a machine, and (3) supervisory models that specify the level of user and machine involvement in supervising the process. Before we proceed to discuss this classification, we shall briefly describe a modeling language, "State Charts", that will allow us to represent these models.

The Finite State Machine Model is a natural medium for describing the behavior of a model-based system. A basic fragment of such a description is a state transition which captures the states, conditions or events, and transitions in a system. The State Chart language is a visual formalism for describing states and transitions in a modular fashion by extending the traditional Finite State Machine to include three unique features: hierarchy, concurrency, and broadcast. Hierarchy is represented by substates encapsulated within a superstate. Concurrency is shown by means of two or more independent processes working in parallel. The broadcast mechanism allows for coupling of components, in the sense that an event in one end of the network can trigger transitions in another. These features of the State Chart are further explained in the following three examples.

(a) Interface models

Figure 13.3 is a modeling structure of an interface model. It has three concurrently active processes (separated by a broken line): speed knob behavior, speed knob indicator, and speed window display. The behavior of the speed knob (middle process) is either "normal" or "pushed-in". (These two states are depicted, in the State Chart language, by two rounded rectangles.) The initial state of the speed knob is normal (indicated by the small arrow above the state), but when momentarily pushed, the speed knob engages or disengages the Speed Intervene submodes of the vertical navigation (VNAV,

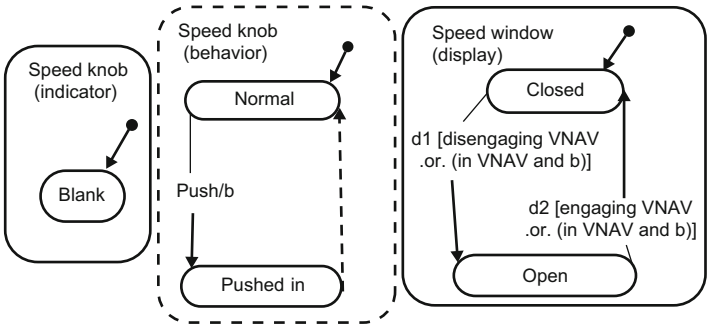


FIGURE 13.3

A model of the structure of the interface model.

hereafter) model. The transition between normal and pushed-in is depicted by a solid arrow and the label "Push" describes the triggering event. The transition back to normal occurs immediately after the pilot lifts his or her finger (the knob is spring-loaded).

The left-most process shown in Figure 13.3 is the speed knob indicator. In contrast to many such knobs that have indicators, for example, the Boeing-757, the speed knob in this example has no indicator and therefore is depicted as a single (blank) state. The right-most process is the speed window display, which can be either closed or open. After VNAV is engaged, the speed window display is closed (implying that the source of the speed is from another component; the flight management computer). After VNAV is disengaged, and a semiautomatic mode such as vertical speed is active, the speed window display is open, and the pilot can observe the current speed value and enter a new one. This logic is depicted in the speed knob indicator process in Figure 13.3: transition d1 from closed to open is conditioned by the event "disengaging VNAV", and d2 will take place when the pilot is "engaging VNAV".

When in vertical navigation mode, the pilot can engage the Speed Intervene submodes by pushing the speed knob. This event, "push" (which can be seen in the speed knob behavior process), triggers event b, which is then broadcast to other processes. Being in VNAV and sensing event b ("in VNAV and b") is another (OR) condition on transition d1 from closed to open. Likewise, it is also the condition on transition d2 that takes us back to close. To this end, the behavior of the speed knob is circular; the pilot can push the knob to close and push it again to open, ad infinitum.

As explained above and illustrated in Figure 13.3, there are two sets of conditions on the transitions between close and open. Of all these conditions, one, namely "disengaging VNAV", is not always directly within the pilot's control; it sometimes takes place automatically (e.g., during a transition from VNAV to the altitude hold mode). Manual re-engagement of VNAV will cause the speed parameter in the speed window to be replaced by economy speed computed by the flight management computer. If the speed value in the speed window was a restriction required by the American Transport Council, the aircraft will now accelerate/decelerate to the computed speed and the American Transport Council speed restriction will be ignored!

(b) Functional models

When we survey the use of models in devices, an additional type emerges: the functional model, which refers to the active function of the machine that produces a distinct behavior. An automatic gearshift

mechanism of a car is one example of a machine with different models, each one defining different behaviors.

As we move to discussion of functional models and their uses in machines that control a timed process, we encounter the concept of dynamics. In dynamic control systems, the configuration and resulting behavior of the machine are a combination of a model and its associated parameter (e.g., speed, time, direction, etc.). Referring back to our car example, the active model is the engaged gear that is Drive, and the associated parameter is the speed that corresponds to the angle of the accelerator pedal (say, 65 miles/h). Both model (Drive) and parameter (65 miles/h) define the configuration of the mechanism. Figure 13.4 depicts the structure of a functional model in the dynamic automated control system of a modern airliner. Two concurrent processes are depicted in this modeling structure: (1) models, and (2) parameter sources.

Three models are depicted in the vertical model superstate in Figure 13.4: vertical navigation, altitude hold, and vertical speed (the default model). All are functional models related to the vertical aspect of flight. The speed parameter can be obtained from two different sources: the flight management computer or the model control panel. The default source of the speed parameter, indicated by the small arrow in Figure 13.4, is the model control panel. As mentioned in the discussion on interface models, engagement of vertical navigation via the model control panel will cause a transition

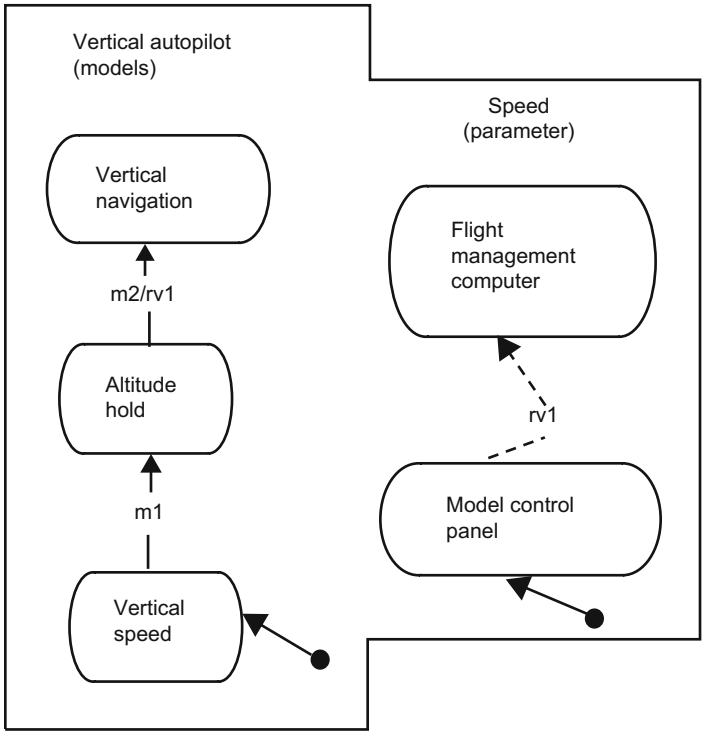


FIGURE 13.4

A model of the structure of the function model.

to the flight management computer as the source of speed. This can be seen in Figure 13.4 where transition m2 will trigger event rv1, which, in turn, triggers an automatic transition (depicted as a broken line) from "model control panel" to "flight management computer". In many dynamic control mechanisms, some model transitions trigger a parameter source change while others do not. Such independence appears to be a source of confusion to operators.

(c) Supervisory models

The third type of model we discuss here is the supervisory model, sometimes also referred to as participatory or control models. Modern automated control mechanisms usually allow the user flexibility in specifying the level of human and machine involvement in controlling the process, that is, the operator may decide to engage a manual model in which he or she controls the process; a semiautomatic model in which the operator specifies target values, in real-time, and the machine attempts to maintain them; or fully automatic models in which the operator specifies in advance a sequence of target values, and the machine attempts to achieve these automatically, one after the other.

Figure 13.5 is an example of a supervisory model structure that can be found in many control mechanisms, such as automated flight control systems, cruise control of a car, and robots on assembly lines. The modeling structure consists of hierarchical layers of superstates, each with its own set of models. The supervisory models in the Automated Flight Control System are organized hierarchically.

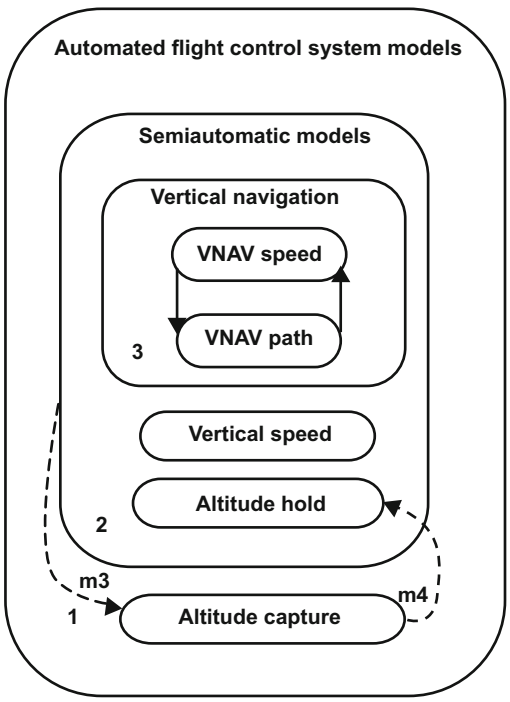


FIGURE 13.5

A model of the structure of the supervisory model.

Three main levels are described in Figure 13.5. The highest level of automation is the vertical navigation model (level 3), depicted as a superstate at the top of the models pyramid. Two submodels are encapsulated in the vertical navigation model; VNAV Speed and VNAV Path; each one exhibiting a somewhat different control behavior. One level below (level 2) are two semiautomatic models: vertical speed and altitude hold.

One model in the automated flight control system, altitude capture, can only be engaged automatically; no direct manual engagement is possible. It engages when the aircraft is beginning the level-off maneuver to capture the selected altitude. When several hundred feet from the selected altitude, an automatic transition from any climb model to altitude capture takes place (m3). In this example, an aspect can be that a transition from vertical navigation or vertical speed to altitude capture takes place (m3). Finally, when the aircraft reaches the selected altitude, a transition back from altitude capture to altitude hold model takes place, also automatically (m4).

In summary, we have illustrated a modeling language, Start Charts, for representing human interaction with control systems, and proposed a classification of three different types of model that are employed in computers, devices, and supervisory control systems. Interface models change display format, functional models allow for different functions and associated parameters, and lastly supervisory models specify the level of supervision (manual, semiautomatic, and fully automatic) in the human machine system. The three types of models described here are essentially similar in that they all define the manner in which a certain component of the machine behaves. The component may be the interface only, a function of the machine, or the level of supervision. This commonality brings us back to our general working definition of the term model; a machine configuration that corresponds to unique behavior.

13.1.2 Systems of human–machine interactions

There are three architectures of human machine systems that are currently popular in industrial control: adaptive, supervisory, and distributed.

(1) Adaptive human machine interface

In a complex control system, the human machine interface attempts to give users the means to perceive and manipulate huge quantities of information under constraints such as time, cognitive workload, devices, etc. The intelligence in the human machine interface makes the control system more flexible and more adaptable. One subset of the intelligent user interface is the adaptive interface. An adaptive interface modifies its behavior according to some defined constraints in order to best satisfy all of them, and varies in the ways and means that are used to achieve adaptation.

In the human machine interface of an industrial control, operators, system, and context are continuously changing and are sometimes in contradiction with one another. Thus, this kind of interface can be viewed as the result of a balance between these three components and their changing relative importance and priority. An adaptive interface aims to assist the operator in acquiring the most salient information, in the most appropriate form, and at the most opportune time. In any industrial control systems, two main factors are considered: the system that generates the information stream and the operator to which this stream is presented. The system and the operator share a common goal: to control the process and to solve any problems that may arise. This common objective makes them cooperate although they may both have their own goals. The role of the interface is to integrate these

different goals with different prioritics and the various constraints that come from the task, the environment, or the interface itself. Specifically, an industrial process control should react consistently, and in a timely fashion, without disturbing the operator needlessly in his task. However the most salient pieces of information should be presented in the most appropriate way.

To develop an adaptive model that meets a set of criteria, the model must: incorporate representations of operator states; represent interface features which are adaptable; represent cognitive processing in a computational framework; provide measures of merit for matches between the input state variables and interface adaptation permutations; operate in a real-time mode; and, finally, incorporate self-evolving mechanisms.

The two main adaptation triggers used to modify the human machine interface could be:

(a) The process: when the process moves from a normal to a disturbed state, the streams of information may become denser and more numerous. To avoid any cognitive overload, the interface acts as a filter that channels the streams of information. To this end, it adapts the presentation of the pieces of information in order to help the operator identify and solve the problem.

(b) The operator: an operator is very difficult to adapt to the user state. As a matter of fact, the interface has to infer whether the operator reacts incorrectly and needs help based on his actions. Then, it may decide to adapt itself to highlight the problem and suggest solutions to assist the user.

The aim of the adaptation is to optimize the organization and the presentation of the pieces of information, according to the state of the process and the inferred state of the operator. What is expected is to improve the communication between the system and the user. The means proposed in an adaptive human machine interface are the following:

(i) Highlight relevant pieces of information. The importance of a piece of information depends on its relevance according to the particular goals and constraints of each of the entities that participates in the communication between the system and the operator.

(ii) Optimize space usage. According to the current usage of the resources, it may turn out that it is necessary to reorganize the display space to cope with new constraints and parameters.

(iii) Select the best representation. According to the piece of information, its importance, the resources available, and the media currently in use, the most appropriate media to communicate with the operator should be used.

(iv) Timeliness of information. The display of a particular piece of information should be timely with respect to the process and the operator. This adaptation should follow the evolution of the process over time, but it should also adapt the timing of the displayed information to the inferred needs of the operator.

(v) Perspectives. In traditional interfaces, the operator has to decide what, where, when, and how the information should be presented. This raises at least four questions: (1) from the client's point of view, is the cost of developing an adaptive human machine interface justifiable? (2) From the human machine interface designer's point of view, is this kind of interface usable and can its usability be evaluted? (3) From the developer's point of view, what are the best technical solutions to implement an efficient system within the required time? (4) From the operators' point of view, is such an adaptive interface a collaborator or a competitor?

(2) Supervisory human machine interface

A supervisory human machine interface can be used in systems where there is a considerable distance between the control room and the machine. It is from this machine house that the controller such as a SCADA or PLC controls the objects, which are, for example, pumps, blowers and purification monitors, etc. To provide the data communication, the supervisory software of the human machine interface is linked with the controllers over a network such as Ethernet, a control area network (CAN), etc. The supervisory software is such that only one person is needed at any one time to monitor the whole plant from a single master device. The generated graphics show a clear representation, on screen, of the current status of any part of the system. A number of alarms are automatically activated directly if parameters deviate from their tight tolerance band. This requires extremely rapid updating of the control room screen contents. All the calculations for the controllers are performed by the control software, using constant feedback from sensors throughout the production process.

In many applications, the supervisory human machine interface is indeed an ideal software package for cases such as the above. Thanks to its interactive configuration and ease of setting up, it is able to get the system up and running and tested in a straightforward manner.

Such an interface follows an open architecture offering all the functions and options necessary for data collection and representation. The system provides comprehensive logging of all measured values in a database. By accessing this database and by using real-time measurements, a wide variety of reports and trend curves can be viewed on the screen or output to a printer.

(3) Distributed human machine interface

The distributed human machine interface is a component-based approach. In a system using such an approach, the interface can directly access any controller component, which also means that each controller displays the human machine interface. Since all the system components are location-transparent, the human machine interface can bind to a component in any location, be it in-process, local, or a remote. The most likely case is remote binding since it is likely that the human machine interface and the controller would reside on different platforms.

In such a set-up, multiple servers are typically used to provide the system with the flexibility and power of a peer-to-peer architecture. Each controller can have its own human machine interface server. A controller's assigned server or proxy server is sufficient for each controller to manage expansion, frequent system changes, maintenance, and replicated automation lines within or across plants.

The primary drawback to decentralized components is the uncertainty of real-time controller performance, generally resulting from poorly designed proxy agent use, for example, if the human machine interface samples controller data at too high a frequency. However, the distributed human machine interface is ideal for SCADA applications, since its distributed peer-to-peer architecture, reusable components, and remote deployment and maintenance capabilities make supporting these applications remarkably efficient. The software's network services have been optimized for use over slow and intermittent networks, which significantly enhances application deployment and communications.

13.1.3 Designs of human–machine interactions

The design of a human machine interface is important, since its effectiveness will often make or break the application. Although the functionality that an application provides to users is important, the way

in which this is provided is equally important. An application that is difficult to use will not be used, hence the value of human machine interface design should not be underestimated.

One of the simplest approaches to interactive systems is to describe the stages that users go through when faced with the task of using it. We can identify roughly five steps for a typical user interaction with a generic interactive system: setting up the design goals, specifying the design principles, executing the design procedures, testing the system performance, and evaluating the design results.

(1) Design principles

The following describes a collection of principles for improving the quality of human machine interface design.

(a) The structure principle. The design should organize the interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users. Putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another are all important. The structure principle is concerned with the overall interface architecture.

(b) The simplicity principle. The human machine interface design should make commonly performed tasks easy to do, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.

(c) The visibility principle. The human machine interface design should keep all needed options and materials for a given task visible, without distracting the user with extraneous or redundant information. Good designs do not overwhelm users with too many alternatives or confuse them with unneeded information.

(d) The feedback principle. The human machine interface design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant to the user through clear, concise, familiar, and unambiguous language.

(e) The tolerance principle. The human machine interface design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.

(f) The reuse principle. The human machine interface design should reuse internal and external components and behaviors, maintaining consistency with purpose, thus reducing the need for users to rethink and remember.

(2) Design process

The design of human machine interfaces should follow these principles and seek a human-centered and integrated automation approach. Functionalities of the interfaces are explained with respect to goals, means, tasks, dialogue, presentation, and error tolerance. The design process itself is considered as a problem-solving activity which contains knowledge about goals, the application domain, human operators, tasks, and human machine interface ergonomics (with the availability of graphical and dialogue editors).

(a) Phase one. The design process begins with a task analysis in which we identify all the stakeholders, examine existing control or production systems and processes, whether paper-based or computerized, and identify ways and means to streamline and improve the process.

Tasks at this phase are to conduct background research, interview stakeholders, and observe people conducting tasks.

(b) Phase two. Once having an agreed-upon objective and set of functional requirements, the next step should generate a design that meets all the requirements. The goal of the design process is to develop a coherent, easy to understand software front-end that makes sense to the eventual users of the system. The design and review cycle should be iterated until a satisfactory design is obtained.

(c) Phase three. The next phase is implementation and testing; along with developing and implementing any performance support aids, for example, on-line help, paper manuals, etc.

(d) Phase four. Once a functional system is complete, we move into the final phase. What constitutes "success" is people using our system, whether it be an intelligent tutoring system or an online decision aid, and are able to see solutions that they could not see before and/or better understand the constraints that are in place. We generally conduct formal experiments, comparing performance using our system with perhaps different features turned on and off, to make a contribution to the literature on decision support and human machine interaction.

(3) Design evaluation

An important aspect of human machine interaction is the methodology for evaluation of user interface techniques. Precision and recall measures have been widely used for ranking non-interactive systems, but are less appropriate for assessing interactive systems. Standard evaluations emphasize high recall levels, but in many interactive settings, users require only a few relevant documents and do not need high recall to evaluate highly interactive information access systems. Useful metrics beyond precision and recall include: time required to learn the system, time required to achieve goals on benchmark tasks, error rates, and retention of the way to use the interface over time.

Empirical data involving human users are time-consuming to gather, and difficult to draw conclusions from. This is due in part to variation in users' characteristics and motivations, and in part to the broad scope of information access activities. Formal psychological studies usually only uncover narrow conclusions within restricted contexts, for example, quantities such as the length of time it takes for a user to select an item from a fixed menu under various conditions have been characterized empirically, but variations in interaction behavior for complex tasks like information access are difficult to account for accurately. A more informal evaluation approach is called a heuristic evaluation, in which user interface affordances are assessed in terms of more general properties and without concern about statistically significant results.

13.2 USER–MACHINE INTERFACES

Modern industrial control systems, either for real-time or for distributed machine control or production automation, typically have a user machine interface. In an industrial control system, programmable controllers provide reliable, real-time machine behavior, while the user machine interface provides the machine operator with a graphical user interface (GUI) for checking the states of machines or devices, monitoring their performance and setting their operating parameters.

13.2.1 User–machine interface system

For all industrial control systems, a high degree of user-friendliness at the interface between human users and machines is necessary in order to be accepted by users. Ambient intelligence applications are characterized by multimodal interfaces, as well as by the proactive behavior of the controller system. Therefore, various interfaces must be sensibly combined with each other, and the interaction with users adapted to the actual operators.

In a typical industrial control system, the user machine interface can be programmed on a PC running Windows, or on a touch panel controller running an embedded operating system such as Windows CE. The user-machine interface features typically include: touch-screen operation; a paged display system with navigation controls; data entry tools (buttons, keypads, etc.); alarm/event displays and logs, etc. The simplest machine control system consists of a single controller running in a "headless" configuration. This configuration is used in applications where a human machine interface is not needed except for maintenance or diagnostic purposes.

(1) User machine interfaces in local control systems

In contrast with the simplest machine control system, the next level of system capability and complexity adds a user machine interface or additional controller nodes. This configuration is typical of machines that are controlled by a local operator. Figure 13.6(A) illustrates this type of configuration.

Most automation companies now offer PC-based solutions for local control systems. The primary concept is to provide a PC-based machine controller that combines multiple technologies from strategic partners such as a human machine interface, motion control, logic functions, machine vision, data acquisition, web access, and other sensors. These are combined with I/O on a single PC platform with a common application programming development environment. All software components in the

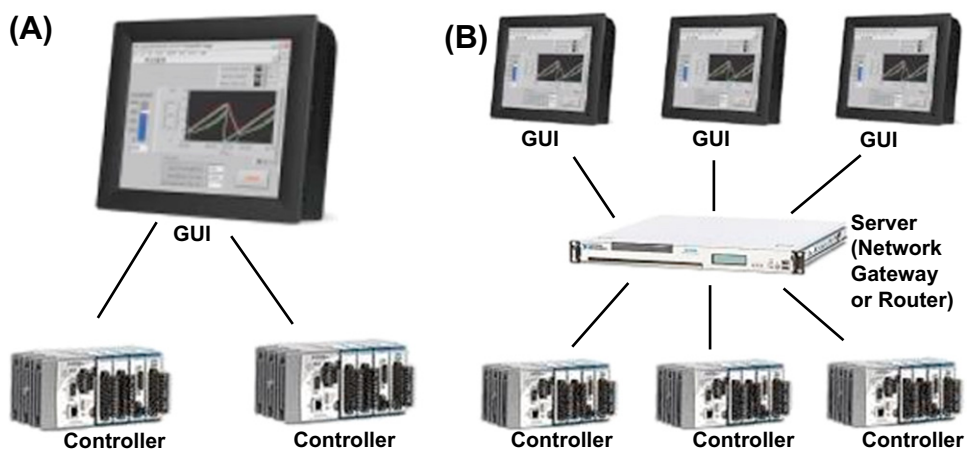


FIGURE 13.6

Industrial control systems with human machine interfaces: (A) a local machine control system; (B) a distributed machine control system.

local control system are pre-tested, and have the necessary drivers installed for further expansion. The controller primarily uses commercially available hardware.

(2) User machine interfaces in distributed control systems

Complex control applications may involve many controllers and human machine interfaces. They often involve a high-end server that acts as a data logging and forwarding engine. This system configuration supports physically large or complex machines, and allows an operator to interact with machines in various locations, or be used to distribute specific monitoring and control responsibilities among a group of operators. Figure 13.6(B) illustrates this type of control system configuration.

Internet-based machine control systems are typical distributed control systems, which offer a new human machine interface solution that uses the wide availability of web browsing software to enable distributed GUI software, remote support and application sharing on the Internet and IP networks. Interactive human machine applications are developed, edited, published and run on a web server, allowing users to easily publish upgrades and applications to remote locations through the use of an Internet connection and a web browser.

As we discussed in Chapter 10, the SCADA system architecture can range from simple local control to a large-scale distributed control system. Therefore, the human machine interfaces in SCADA systems can range from PC-based to network-based, and even to Internet-based.

All human machine interfaces are available with a variety of devices. Devices that provide real-time clock support use a special battery and are not connected to the power supply. Power-over-Ethernet (PoE) equipment eliminates the need for separate power supplies altogether. Human machine interfaces that offer shielding against electromagnetic interference and radio frequency interference are commonly available. Devices that are designed for harsh environments include enclosures that meet standards from the National Electronics Manufacturers' Association (NEMA).

(1) Operator interface terminals

These human machine interfaces are terminals that users interact with in order to control other devices. Some have knobs, levers, earphones, and screens. Others provide programmable function keys or a full keypad. Devices that include a microprocessor or interface to personal computers are also available. Many human machine interfaces include alphanumeric or graphical displays. For ease of use, these displays are often backlit or use standard messages. When selecting human machine interfaces, important considerations include the range of devices that can be supported and controlled. Device dimensions, operating temperature, operating humidity, and vibration and shock ratings are other important factors.

Traditional human machine interfaces have used light-emitting diodes (LEDs) for flat panel displays. In LEDs, the diode is such that light emitted at a p n junction is proportional to the bias current; color depends on the material used. However, many human machine interfaces now include flat panel displays which use liquid crystal display (LCD) or gas plasma technologies. In LCD, an electric current passes through a liquid crystal solution that is trapped between two sheets of polarizing material. The crystals align themselves so that light cannot pass, producing an image on the screen. LCD can be monochrome or color. Color displays can use a passive matrix or an active matrix. Passive matrix displays contain a grid of horizontal and vertical wires with an LCD element at each intersection. In active matrix displays, each pixel has a transistor that is switched directly on or off,

improving response times. Unlike LCD, gas plasma displays consist of an array of pixels, each of which contains red, blue, and green subpixels. In the plasma state, gas reacts with the subpixels to display the appropriate color.

These human machine interfaces differ in terms of performance specifications and I/O ports. Performance specifications include processor type, random access memory (RAM), hard drive capacity, and other drive options. I/O interfaces allow connections to peripherals such as mice, keyboards, and modems. Common I/O interfaces include Ethernet, Fast Ethernet, RS-232, RS-422, RS-485, small computer system interface (SCSI), and universal serial bus (USB). Ethernet is a local area network (LAN) protocol that uses a bus or star topology and supports data transfer rates of 10 Mbps. Fast Ethernet is a 100-Mbps specification. RS-232, RS-422, and RS-485 are balanced serial interfaces for the transmission of digital data. Small computer systems interface (SCSI) is an intelligent I/O parallel peripheral bus with a standard, device-independent protocol that allows many peripheral devices to be connected to the SCSI port. Universal serial bus (USB) is a four-wire, 12-Mbps serial bus for low-to-medium speed peripheral device connections.

(2) Operator interface monitors

Machine controllers and monitors use electronic numeric control and a monitoring interface for programming and calibrating computerized machinery. This product area includes general-purpose machine controllers, embedded machine controllers, machine monitors, CNC stepper motors, and CNC router controllers. A machine controller is a programmable, automatic, and computer numerically controlled (CNC) device. An embedded machine controller is part of a larger system. A machine monitor is used to collect and display machine or system data from devices or equipment. For example, a CNC stepper motor is used to drive a machine tool with power and precision; a CNC router controller is used to cut tool paths. Many other types of machine controllers and monitors are also available.

Machine controllers and monitors consist of many different components, that normally use a microprocessor to perform predetermined control and logical operations. Memory is added to the microprocessor in order to record data from the machine, and an input device is used to provide menus or options. Machine monitors track a machine's uptime, downtime, and idle time and allow operators to enter a reason for downtime or nonproductive activities. In some cases, a machine monitor can be programmed to require the entry of a reason code after each downtime event. In this way, machine controllers and monitors can be configured to meet the needs of specific machinery and industries.

They are used in many different applications, for example, to regulate medical equipment such as respirators. Others are used in aerospace, automotive, or military applications. An embedded machine controller can be used in a printing machine, pipe-bending equipment, CNC stepper motor, or CNC router controller. They are also used in the manufacture of semiconductors and electronic devices. Machine controllers and monitors with integral software are used in industries where reliability, quality, and cost are important considerations.

(3) Industrial control pendants

Industrial control pendants are sophisticated, hand-held terminals that are used to control robot or machine movements from point to point, within a determined space. They consist of a hanging control console furnished with joysticks, push buttons, or rotary cam switches. Teach pendants, a type of industrial control pendant, are the most popular robotics teaching method, and are used in many industries. As the robot moves within its determined space, various points on its path are recorded into

its memory banks, and can be located later on through subsequent playback. There are a number of teach pendant types available, depending on the application type. If the goal is simply to monitor and control a robotics unit, then a simple control box style is suitable, but if additional capabilities, such as on-the-fly programming, are required, more sophisticated boxes are available.

Industrial control pendants are equipped with switches, dials, and push-buttons through which data are relayed to the robotics unit, and to additional monitoring systems if necessary. The relationship between pendants and their subservient unit is generally established via an interconnected cabling system, but wireless devices are also available.

During use, the operator actuates the switches on the manual pendants in a specific order. This causes the robot, end effectors, or machine, to move to and from the desired points. As the end effector reaches the desired point, the operator uses the record push-button to enter the location into memory. This is the most common programming method for playback robots.

Control pendants do have a significant disadvantage, in that the operator must divert his or her attention away from the movement of the machine during programming in order to locate the appropriate push-button to move the robot. The use of a joystick provides a solution to this problem as the movement of the stick sends the robot or machine in that direction.

13.2.2 User–machine interface hardware

The user machine interface hardware consists of a number of integrated-circuit modules, known as human machine interface controllers in industrial markets. They provide various GUI control modules including keypad control, LCD or LED control, black-white brightness control, color mixing, tone generation, as well as software translation, etc. They are often integrated into a chipset card, known as the human machine interface daughter card. Figure 13.7 shows a photograph of such, and its function module diagram.

Highly interactive GUIs require more powerful microprocessors. Unfortunately, many on the market today do not meet the performance characteristics necessary to deliver a compelling, interactive, animated GUI; nor do they include on-chip support for peripherals, graphics acceleration, and LCD displays, making the total package a very expensive solution if assembled from discrete components.

In contrast, FPGA devices, composed of an array of logic cells that can be configured to perform a variety of functions, make a better choice due to their better performance and flexibility. When combined with an embedded software core, FPGA devices easily support both the microprocessors' general processing functions and the functionalities of other external devices. They offer unprecedented scalability, because they can be adapted dynamically for different screen sizes, image resolutions, peripherals, and GUIs.

Design examples with a FPGA device for four control modules on the daughter card from Figure 13.7 are given in Figure 13.8. They offer innovative ideas for FPGA applications and help users create designs that utilize their advantages.

(a) Keypad control module (Figure 13.8(A)). This module senses the keypad (a standard cellphone keypad with 18 keys, for example) by scanning the 6 rows and reading the 3 columns. This design can be targeted to any application requiring a keyboard interface in a matrix form. The row-column matrix combination can easily be changed to cater for the designer's requirements. The scan method can also be integrated into an interrupt-based control logic function when interfacing to a microprocessor.

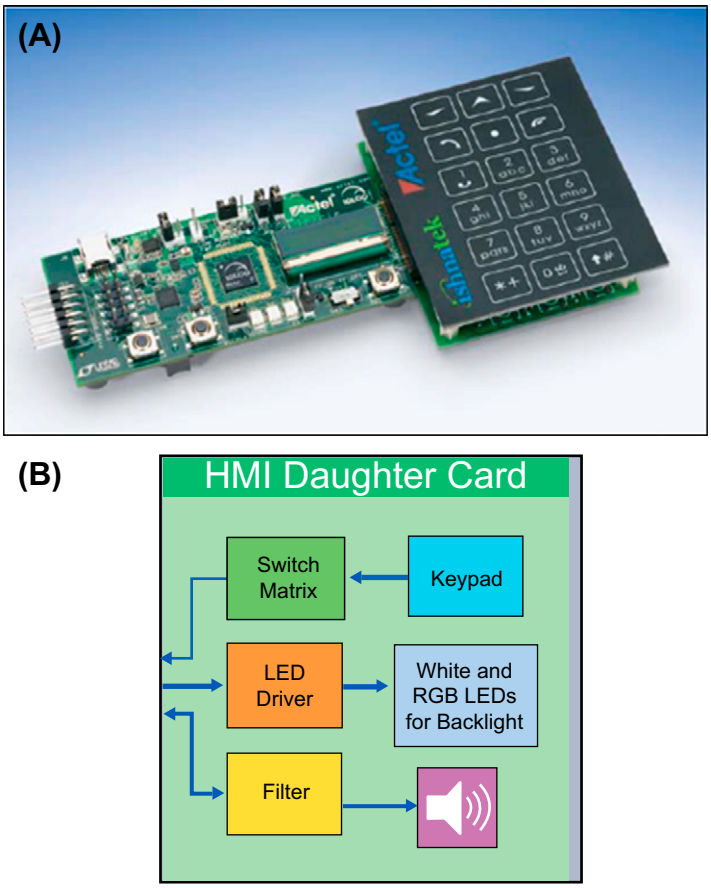


FIGURE 13.7

A daughter card for a human machine interface: (A) product photo; (B) function module diagram.

(Courtesy of the Actel Corporation, 2009.)

(b) Brightness control for white LED ((Figure 13.8(B)). This module controls the intensity and brightness of white LED (WLED) by varying the duty cycle of the pulse-width-modulation (PWM) logic. The 8-bit (256 steps) PWM drives the LEDs through a WLED driver chip.

(c) Color mixing for red-green-blue (RGB) LEDs ((Figure 13.8(C)). This module controls the color mixing for the RGB LEDs using three PWM signals. This scheme can be used to generate keypad backlight or LCD backlight of any color, or to illuminate a particular area with required color using the RGB LEDs. The brightness and color of the RGB LEDs are controlled through three 8-bit PWMs. These PWM signals are time-division multiplexed to reduce power consumption.

(d) Tone generation ((Figure 13.8(D)). This design generates tones of desired frequency (period) and volume (duty cycle) using a 16-bit PWM signal. Tone duration and dampening can be controlled through additional counter logic.

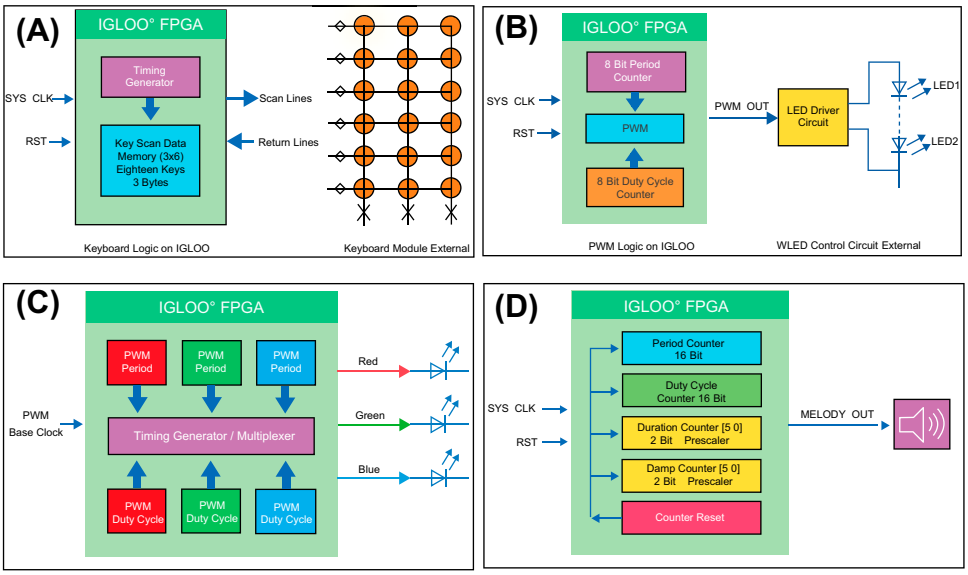


FIGURE 13.8

Four control modules on the daughter card of a human machine interface in Figure 13.7: (A) keypad control module; (B) brightness control module; (C) color mixing control module; (D) tone generation control module.

(Courtesy of the Actel Corporation, 2009.)

Of course, a GUI control module can be made as an independent subsystem. Figure 13.9 provides a high-level, hierarchical view of the peripherals and interfaces that implement an LCD controller design. The video pipeline in FPGA, the LCD touch-screen module, and the CPLD (complex programmable logic device) are the major components of the LCD controller. The video pipeline is responsible for driving data signals on the LCD module data bus and for reading frame buffer data generated by the microprocessor. A series of specialized peripherals allows data units to be converted between buses with different widths, in this case, a 24-bit RGB pixel input stream to an 8-bit pixel output stream in which each RGB color component is transmitted separately. The video synchronizing generator peripheral transmits pixel data to the LCD touch-screen module by sequencing the control and data signals for the data bus of the module.

13.2.3 User–machine interface software

A human machine interface also requires software to enable users to manage industrial and process control machinery via a GUI on devices such as operator terminals and monitors.

There are two basic types of human machine interface: supervisory-level and machine-level. The first is designed for control room environments and used for state display, data acquisition and control operations, whilst the second uses embedded, machine-level devices within the machine or device itself. Most interface software is designed for one or the other, but some that are suitable for both types

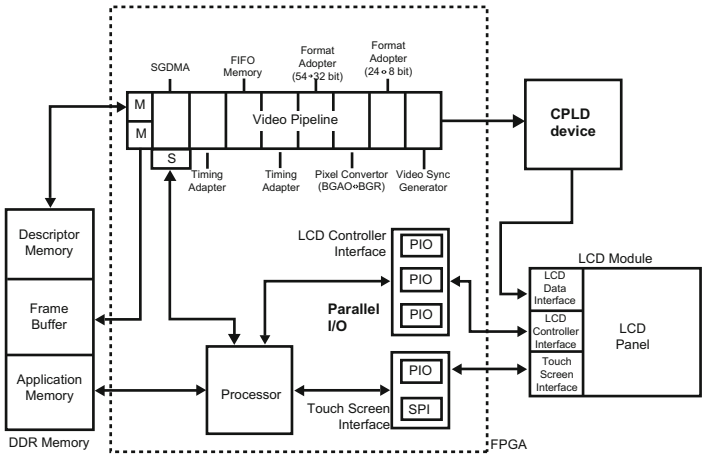


FIGURE 13.9

A function block diagram for a LCD control subsystem of a human machine interface on a FPGA device.

are also available. For example, a human machine interface program, FactoryTalk View, has been developed by Rockwell Automation using the Rockwell Automation Integrated Architecture. FactoryTalk View is part of a scalable and unified suite of monitoring and control solutions designed to support both machine-level and supervisory-level human machine interface applications. FactoryTalk View includes FactoryTalk View Machine Edition and FactoryTalk View Site Edition.

FactoryTalk View Machine Edition is a machine-level human machine interface that supports both open and embedded operator interface solutions for monitoring and controlling individual machines or small processes. It allows for a consistent operator interface across multiple platforms, including Microsoft Windows. Its components are a PC-based development tool called FactoryTalk View Studio and a separate run-time system called FactoryTalk View Machine Edition Station.

FactoryTalk View Site Edition is a human machine interface for supervisory-level monitoring and control applications. It has a distributed and scalable architecture that supports distributed client/server applications, giving maximum control over information where users want it. This highly scalable architecture can be applied to a stand-alone one-server/one-client application, or to multiple users interfacing with multiple servers. FactoryTalk View Site Edition is targeted at applications that need a distributed and scalable architecture, including run-time servers and clients, so allowing customers to develop and deploy multiple-server/multiple-client applications. Such applications are developed with the FactoryTalk View Studio development tool.

(1) Typical human machine interface software library modules

Most human machine interface software is packaged into library modules. The following lists some useful ones.

(a) The graphics monitoring module lets developers add display objects to the application to make the operator interface as intuitive as possible, thereby decreasing training and support requirements.

This module includes up to 256 color support and free-form graphics, giving developers the flexibility to import pictures and animate images to create an application that is easy to use.

(b) The machine configuration module is designed to address specific requirements of machine builders and users. With this module, software developers have the ability to design, manage and perform efficient manufacturing changeovers. This module also makes it possible to develop modular configurations and machine set-up functions rapidly, making it easy to customize product.

(c) The data transfer module provides a high-speed gateway for sharing information between different types of control equipment by using multiple drivers. This module allows replacing expensive networking software and hardware with inexpensive software connections.

(d) The historical trending module gathers selected data from the machine or process, to be viewed online in graphical or spreadsheet format. These data files can also be logged to disk so that they can be analyzed offline. Data can be logged on time interval, event or operator input. Advanced features such as data and time search allow for enhanced on-line analysis of logged data.

(e) Networking module: this module enables the transfer of multiple applications throughout the manufacturing facility or in remote locations using off-the-shelf LAN/WAN operating systems. It is a "network aware" architecture that simplifies centralization of applications and/or program files on a network server.

(f) The report module creates free-form reports for the control system application. These real-time data reports can be previewed on the display screen while in online operation; completed reports can be downloaded to a network or disk for review at a later date. Reports can also be printed in hard copy or ASCII file format.

(2) Typical human machine interface software system architecture

A human machine interface is built on either a microprocessor or an FPGA chipset, which requires its software to include at least an application program and an operating system. Figure 13.10 shows a typical human machine interface software system architecture. Obviously the human machine interface software must be compatible with the system's operating system and hardware.

It is clear that a human machine interface is one of the components of an industrial control system, so it should have a data transfer or network module for communication with other components of the system.

13.3 INDUSTRIAL APPLICATION EXAMPLES

Over recent years, there have been many developments in industrial control and automation systems which have led to human machine interfaces becoming a necessary component of most industrial control and automation systems. Two important examples in this respect are the human robot interfaces in industrial robots, and the human machine interfaces in SCADA systems.

13.3.1 Human–machine interfaces in robotic systems

A human machine interface in a robotic system is a terminal that allows the human operator to control, monitor, and collect data, and can also be used to program the system.

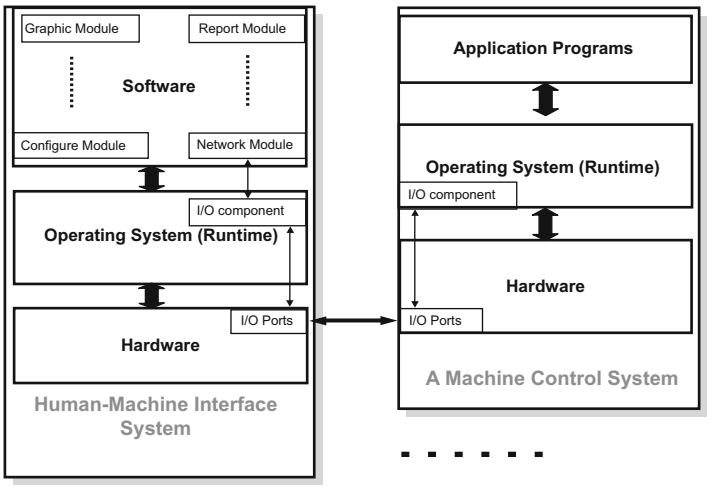


FIGURE 13.10

Typical architecture of a human machine interface software system.

Sometimes the human machine interface can be built into the mainframes of the robotic systems. This kind of human machine interface can be a computer screen, pendant consoles, spin buttons, and arm sticks. However, the interface may also go through a chain of devices that on one side convert human-intelligible commands into robot commands and on the other side convert robot feedback into human-understandable information. At the beginning of the chain (close to the human operator) there is a robot control station equipped with a series of human machine interfaces, at the other end there is a robot controller.

Different human machine interfaces are needed to deal with different types of involvement of the human operator. For programming activities, the consolidated human machine interfaces of computer consoles are adequate; for advanced programming involving interaction with three-dimensional computer models, joysticks and a space-mouse could be used; for limited teleoperation, a set of joysticks has traditionally been used; for advanced and immersive teleoperation with force feedback, master-arms and Exoskeletons are the best choice

While performing any robot operation, a human operator may interact with the robot in degrees ranging between two extremes: no interaction and constant interaction. The two control models associated with these extremes are:

1. Offline robot programming, in which everything that the robot is asked to do is encoded in a program (in a suitable task-description language). This program is to be downloaded via proprietary interface equipment. Available equipment for industrial robot offline programming include proprietary computers (PC, desktop, and laptop computers), handheld terminals, and so on. Figure 13.11 illustrates the model for industrial robots.
2. Online robot programming, in which a human operator drives (or teaches) the robot through its tasks by means of suitable human computer interface equipment. An operator doing the teaching has physical contact with the robot's human computer interface or arms to actually

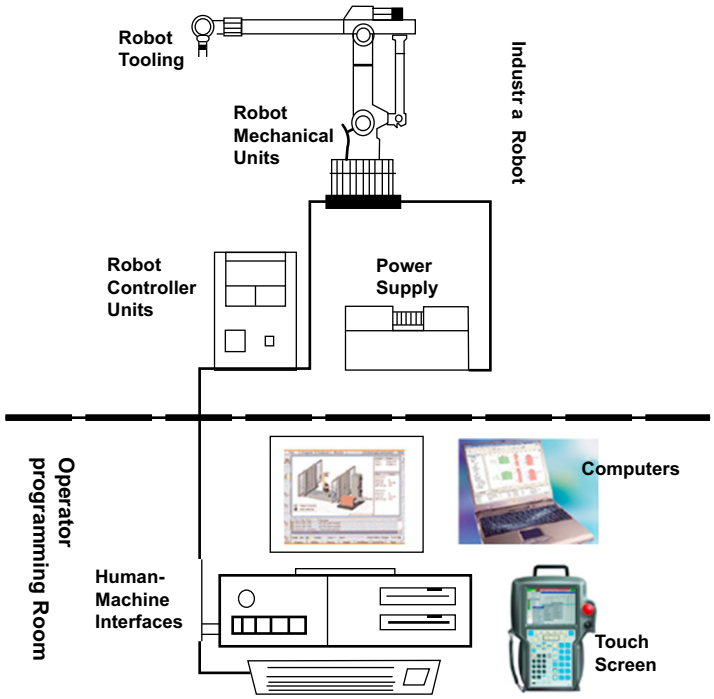


FIGURE 13.11

Industrial robot offline programming: with offline programming the required functional and positional steps are written on remote human machine interface equipment (such as computer consoles, touch screens, etc.), then electronically transferred to the robot controller unit.

gain control or perform the teaching functions within the working envelope. The human machine interface equipment in the market for industrial robot online programming include teach pendants, robot arms, and joysticks or space-mouse for teleoperation. Figure 13.12 illustrates the online programming model for industrial robots.

Offline programming is suited to applications where the same operation is repeated in a well-known non-changing environment, whereas online programming suits applications where variation of the operation and an unknown or unpredictable environment are present, hence requiring continuous human judgement.

13.3.2 Human–machine Interfaces in SCADA systems

The human machine interface is an essential component of any SCADA system, to perform supervisory controls and data acquisition. It presents the information to the human operator graphically, mostly in the form of mimic diagrams. Mimic diagrams may consist of line graphics and schematic symbols to represent process elements, or may consist of digital photographs of the process equipment

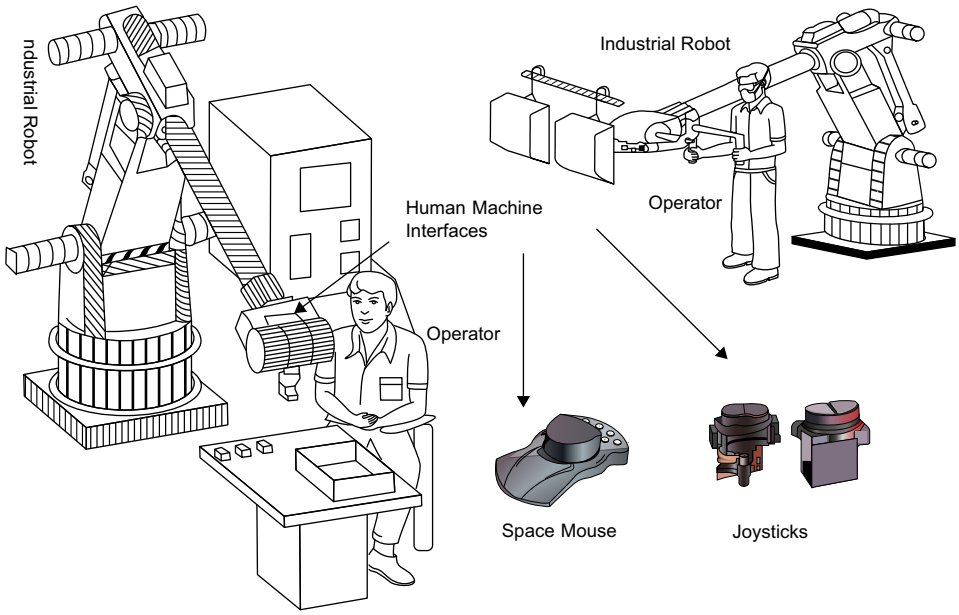


FIGURE 13.12

Industrial robot online programming: online programming uses some proprietary human machine interface equipment (such as teach pendants, joysticks, space-mouse, etc.), with which the human operators can physically teach the robot through the desired sequence of events by activating the appropriate buttons or switches. The new programs with the required position and functional information are "taught" to the robot controller unit.

overlain with animated symbols. This means that the operator can watch a schematic representation of the devices and processes being controlled so as to monitor them efficiently.

A human machine interface of a SCADA system is the graphic apparatus, which mainly refers to computer screens and TV consoles, but can also be specially built industrial programmable controllers such as CNC (computer numerical controller), or PLC (programmable logic controller) devices, as shown in Figure 13.13. The interface is usually linked to the databases and software of a SCADA system, to provide status, trending, diagnostic data, and management information.

The human machine interface package for a SCADA system typically includes a drawing program to allow operators to change the way these points are represented in the interface. These representations can be as simple as an on-screen traffic light, which represents the state of an actual traffic light in the field, or as complex as a multiple-projector display representing the position of all of the elevators in a skyscraper or all of the trains on a railway.

In water treatment plants, where tanks, lagoons, pumping stations and other equipment are spread over a wide area, camera images can make sure that valves open and close, lagoons are at the right level, and a child has not fallen into the lagoon. Video not only provides process information, it can mitigate liability and provide security. With these videos, a human operator at a central control room can know whether to take a shotgun or a wrench to the field to fix a stuck valve.

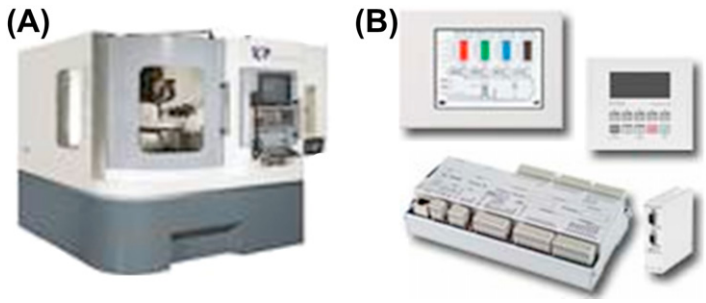


FIGURE 13.13

Some examples of industrial controllers with built-in human machine interfaces: (A) a CNC (computer numerical controller) with graphical and non-graphical user interfaces; (B) a PLC (programmable logic controller) with graphical and non-graphical user interfaces.

Figure 13.14 shows a human machine interface screen at a water treatment plant. If a video image of the process unit appeared on the screen when an exception occurred, the human operator at a central control room would be able to immediately see what was happening.

At present, many SCADA networks are almost hack-proof, accident-resistant and control-reliable because of their human machine interfaces. Figure 13.15 shows a SCADA network designed to handle

FIGURE 13.14

A graphic screen in the central control room of a water treatment plant, which allows the human operator to quickly glance at video images from important areas of the water treatment process.

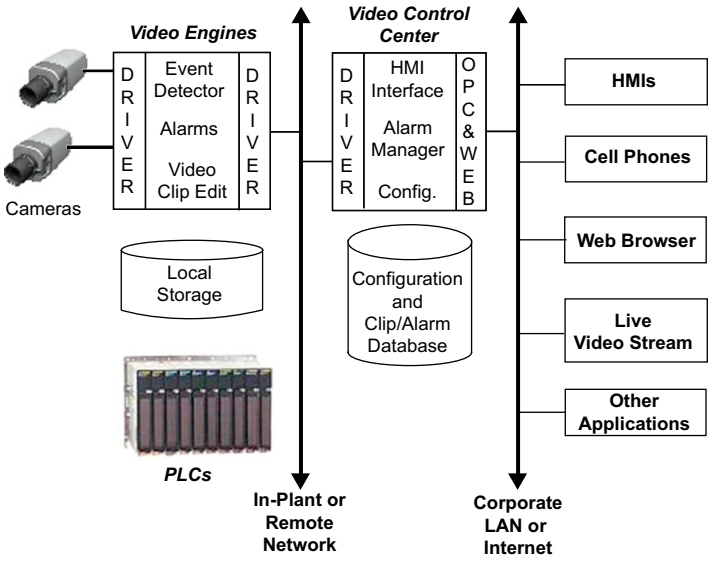


FIGURE 13.15

This diagram shows how cameras transmit data over standard industrial SCADA networks to a video processor. The video processor stores the data in a relational database and makes images available to a SCADA system with human machine interfaces (HMIs), DCS, PLC, process databases, website browsers, and cellphones, etc.

(Courtesy of Steve Rubin, Longwatch Inc., in 2009.)

digital communications between industrial controllers and computers, which can easily accommodate video images, too. The cameras use the bandwidth available in most SCADA networks to report video information to the host system for real-time diagnostics and handles.

Problems

1. Think about whether the gear lever, brake pedal, clutch pedal, and steering wheel can be included in the "human machine interface" of automobiles.
2. Based on Figure 13.1, specify the corresponding components on both sides of the human machine interaction involved between the driver and the car.
3. Figure 13.2 shows an operator model in which their cognitive behavior and action mode differ in accordance with their perception of the information presented at the interface. Can you work out a system model in which the system's production performance and control configurations differ in accordance with the system's demonstration of the information presented at the interface?
4. Please set up your "human machine interaction" model between a driver and a car by following these steps: (1) set up a components states table in accordance with Table 13.1; (2) select one of these components, set up its "interface model", "function model" and "supervisory model", and plot their Finite State Machine diagrams, respectively.
5. By taking either a desktop or a laptop personal computer as an example, please specify the role of the human operator in an adaptive human machine interface, and list all the hardware and software requirements for an adaptive human machine interface system.

6. Please classify these human machine interface architectures: (1) the operator interface for controlling a railway track fork at a railway station; (2) the operator interface at a central control room of a city traffic control system; (3) flight information screens at an airport.
7. The Color Mixing Module in Figure 13.8(C) controls the color mixing for the red green blue LEDs using three PWM signals. The brightness and color of the red green blue LEDs are controlled through three 8 bit PWMs. Please analyze why these PWMs signals are time division multiplexed.
8. Based on your understanding of the graphical user interfaces of a multifunctional office equipment or copier, and a desktop or laptop PC computer or other industrial equipment, plot their hardware module (components) and software module (components) diagrams. (Please note that, in general, there is one to one correspondence between hardware modules (components) and software modules (components) in a microprocessor unit or chipset).
9. If possible, get a computer installed with an MS DOS operating system and a computer installed with a Windows XP operating system, compare their GUI terminals (monitors), and write an essay listing the differences between their GUI terminals (monitors).
10. If you are familiar with the Microsoft Visual Studio, use C#, C++ or C to plot some graphical user interfaces like those on the devices you can get from a laboratory, workshop, or office.
11. Explain the purpose of robot programming. Name the two major categories of robot programming and compare their advantages and disadvantages.
12. Explain why a human machine interface is an essential for any SCADA systems to perform supervisory control and data acquisition.
13. An urban traffic surveillance system is a typical SCADA system. You can watch the city road traffic status on a TV channel. Please explain whether or not the TV channel showing a city road traffic status is a human machine interface for this SCADA system.
14. Alarms can be created in such a way that when their requirements are met, they are activated. An example of an alarm is the "fuel tank empty" light in a car. Please investigate some alarm solutions in a SCADA's human machine interfaces.

Further Reading

Andrew Sears, Julie Jacko (Eds.). The Human Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications (2nd edition). CRC Press. 2007.

F. Wallhoff, M. Ablaßmeier et al. Adapative Human machine Interfaces in Cognitive Production Enviroments. Proceedings of the International Congress on Mathematical Education (ICME; 2006). pp. 2246 2249. 2007.

Schneider Electric (http://www.automation.schneider electric.com). Human machine interface. http://www.automation.schneider electric.com/as guide/EN/pdf files/asg 8 human machine interface.pdf. PP14. Accessed: September 2009.

Asaf Degani, Michael Shafto, Alex Kirlik. Modes in Human machine Systems: Review, Classification, and Application. International Journal of Aviation Psychology, 9(2), 125 138. 2000.

Human Engineering Limited (http:// www.humaneng.co.uk). Human systems interface design capability statement. http://www.humaneng.co.uk/assets/capstats/HEA%20HCI%20Capstat.pdf. Accessed: September 2009.

Altera Corporation (http://www.altera.com). White paper (WP 01083 2.0; July 2009): Implementing a Cost Effective Human machine Interface. 2009.

Hidekazu Yoshikawa. Human machine interaction in nuclear power plant: http://article.nuclear.or.kr/jknsfile/v37/JK0370151.pdf. Accessed: September 2009.

GlobalSpec (http://www.globalspec.com/). Human machine interfaces. http://www.globalspec.com/MyGlobalSpec/Comp%3D2248&areaId=2248. Accessed: October 2009.

GlobalSpec (http://www.globalspec.com/). Product categories for man machine interface. http://industrial computers.globalspec.com/Industrial Directory/graphics card. Accessed: October 2009.

Mauro Marinilli. The theory behind user interface design. http://www.developer.com/design/article.php/10925 1545991. Accessed: October 2009.

National Instruments (http://www.ni.com). Choosing a machine control architecture. http://zone.ni.com/devzone/ cda/tut/p/id/6105. Accessed: October 2009.

Actel (http://www.actel.com). Human machine interface solution. http://www.actel.com/products/solutions/hmi/ default.aspx. Accessed: October 2009.

Actel (http://www.actel.com). HMI daughter card. http://www.actel.com/products/solutions/hmi/dcard. aspx#overview. Accessed : October 2009.

Parker Hannifin Corporation (http://www.compumotor.com). Human Machine Interface Software. Catalog 8000 4/USA. 2009.

Rockwell Automation (http://www.rockwellautomation.com). FactoryTalk View. http://www. rockwellautomation.com/rockwellsoftware/performance/. Accessed: October 2009.

Wikipedia (http://en.wikipedia.org). SCADA. http://en.wikipedia.org/wiki/SCADA. Accessed: October 2009.

Epson Robot (http://www.robots.epson.com). Robot case studies. http://www.robots.epson.com/general/ casestudies.htm#IndustrialPipe. Accessed: October 2009.

Automation.com (http://www.automation.com). HMI and SCADA White Papers. http://www.automation.com/ enews/hmi scada/HMI2008 March.htm. Accessed: October 2009.

Steve Rubin. Video process monitoring. http://www.automation.com/resources tools/articles white papers/hmi and scada software technologies/video process monitoring. Accessed: October 2009.

U.S. Department of Labor, Occupational Safety & Health Administration. Industrial Robot and Robot System safety. http://www.asimo.pl/dodatki/robot system safety.pdf. Accessed: October 2009.

CHAPTER

Data transmission interfaces 14

The term data transmission concerns the transmission of digital electric or electromagnetic signals from source to destination through some electric media over a physical distance, whereas analog transmission involves the transmission of analog signals. The primary difference between the two is whether the transferred and processed messages in the system are digital or analog.

The digital signal refers to two concepts. It can refer to discrete-time signals that have a discrete number of levels, for example a sampled and quantified analog signal; or to the continuous-time waveform signals in a digital system that represent a bit-stream. In the first case, a signal that is generated by means of a digital modulation method is considered to be converted from an analog signal, while it is considered as a digital signal in the second case.

An analog signal is a continuously variable signal in both time and amplitude, generally carried by use of modulation. Quantization of information is not necessary, unlike digital signals. Data are represented by physical quantities rather than digital bits. Analog transmission enables information to be transmitted from point-to-point or from one point to many. Once the data have arrived they are converted back into digital form to be processed by the receiving device.

Any industrial control system (or network) connects equipment such as computers, programmable controllers, robots, and printers, which can be the terminals for transmitting and receiving data signals. These terminals are called DTEs (data terminal equipment). However, the electrical signals transmitted across the cables or wires must be analog. This requires conversion between digital and analog signals and vice versa. The converters are modems or multiplexers, known as DCEs (data circuit-terminating equipment).

DTEs can be very diverse in electrical parameters, mechanical properties and transmission modes, etc. For example, some computers or programmable controllers use serial ports, whereas some use parallel. However, there is no such diversity in DCEs. To solve conflicts between DTEs and DCEs, special interfaces have been designed as the DTE DCE Interface Standards. Typical DTE DCE Interface Standards are RS (Recommended Standard) series such as RS-232, RS-485, etc.

Data transmission between DTEs must be either synchronous or asynchronous, so any DTEs must have special devices to handle synchronous or asynchronous receiving and transmitting. These special devices are termed transmission control devices.

Any industrial control system consists of DTE, DCE, DTE-DCE interface devices, and transmission control devices. These elements can be configured as shown in Figure 14.1. This chapter briefly introduces these elements as they are used in industrial control systems.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10014-2

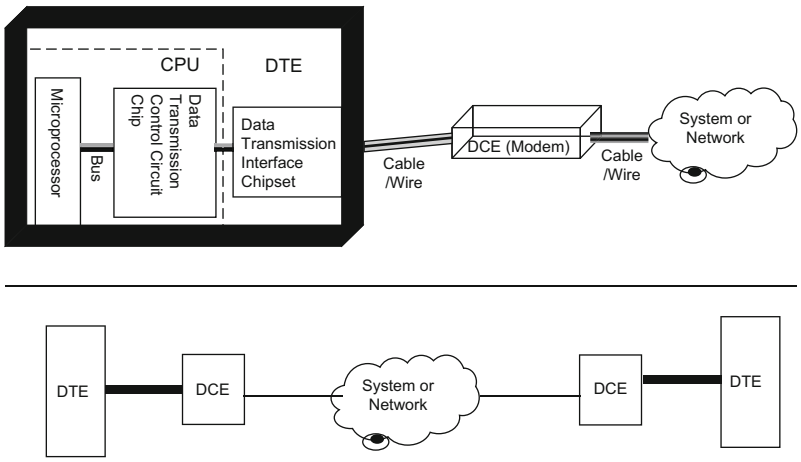


FIGURE 14.1

A systemic configuration of data transmission interfacing and controlling devices.

14.1 DATA TRANSMISSION BASICS

All digital electric or electromagnetic information must be transmitted by means of at least a transmission channel, which is a pathway for conveying the information. It may be defined by an electrical wire that connects devices, or by a radio, laser, or other wireless energy source that has no obvious physical presence. Information sent through a transmission channel has a source from which the information originates, and a destination to which it is delivered. Although originating from a single source, there may be more than one destination, depending upon how many receiving stations are linked to the channel and how much energy the transmitted signal possesses.

A digital message is represented by an individual data bit, which is one digit in the binary numbering system, either 0 or 1. In data transmission, bits can be encapsulated into multiple-bit message units. A byte, which consists of eight bits, is an example of a message unit that may be conveyed through a digital transmission channel. A collection of bytes may itself be grouped into a form of message or other higher-level message unit, such as character, word, or page.

In practice, most digital messages are vastly longer than just a few bits, and therefore the whole cannot be transmitted instantaneously. There are two ways to transmit the bits that make up a data message: serial or parallel. Serial transmission sends bits sequentially, one bit at a time, over a single channel. In parallel transmission, bits of a data message are transmitted simultaneously over a number of channels, which are generally organized in multiples of a byte. Figure 14.2 is an illustration of bit-serial and bit-parallel transmissions, respectively.

Serialized data are not generally sent at a uniform rate through a channel. Instead, there is usually a burst of regularly spaced binary bits followed by a pause, after which the data flow resumes. Packets of binary data are sent in this manner, possibly with variable-length pauses between packets, until the

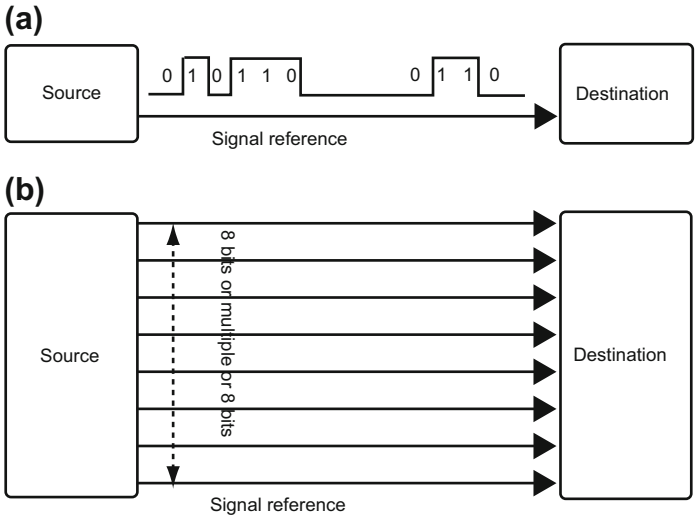


FIGURE 14.2

Illustrating: (a) bit-serial transmission, (b) bit-parallel transmission.

message has been fully transmitted. In order for the receiving end to know the proper moment to read individual binary bits from the channel, it must know exactly when a packet begins and how much time elapses between bits. Two basic techniques, synchronous and asynchronous techniques, are employed to ensure correct serial data transmission.

(1) In synchronous systems, separate channels are used to transmit data and timing information. The timing channel transmits clock pulses to the receiver. Upon receipt of a clock pulse, the receiver reads the data channel and latches the bit value found on the channel at that moment. The data channel is not read again until the next clock pulse arrives. Because the transmitter originates both the data and the timing pulses, the receiver will read the data channel only when told to do so by the transmitter (via the clock pulse), and thus synchronization is guaranteed.

(2) In asynchronous systems, a separate timing channel is not used. The transmitter and receiver must be preset in advance to an agreed-upon baud rate. The baud rate refers to the rate at which data are sent through a channel, and is measured in electrical transitions per second (at bits per second: bps). A very accurate local oscillator within the receiver will then generate an internal clock signal that is equal to the transmitters within a fraction of a percent. For the most common serial protocols, data are sent in small packets of 10 or 11 bits, eight of which constitute message information, while the others are used for checking. When the channel is idle, the signal voltage corresponds to a continuous logical "1". A data packet always begins with a logical "0" (the start bit) to signal to the receiver that a transmission is starting. This triggers an internal timer in the receiver which generates the necessary clock pulses. Following the start bit, eight bits of message data are sent bit by bit at the agreed baud rate. The message is concluded with a parity bit and a stop bit.

Two codeword standards are used for both bit-serial and bit-parallel transmissions. Extended Binary Coded Decimal Interchange Code (EBCDIC) Standard is a binary coding scheme developed by IBM Corporation for the operating systems within its larger computers. EBCDIC is a method of assigning binary number values to characters (alphabetic, numeric, and special characters such as punctuation and control characters).

American Standard Code for Information Interchange (ASCII) Standard is a single-byte encoding system (i.e., uses one byte to represent each character), and using the first seven bits allows it to represent a maximum of 128 characters. ASCII is based on the characters used to write the English language (including both upper and lower-case letters). Extended versions (which utilize the eighth bit to provide a maximum of 256 characters) have been developed for use with other character sets.

Both electrical noise and disturbances may cause data to be changed as they pass through a transmission channel. If the receiver fails to detect this, the received message will be incorrect, possibly resulting in serious consequences. A parity bit is added to a data message for the purpose of error detection. In the even-parity convention, the value of the parity bit is chosen so that the total number of "1" digits in the combined data plus the parity packet is an even number. Upon receipt of the message, the parity needed for the data is recomputed by local hardware and compared to that received with the data. If any bit has changed state, the two will not match, and an error will have been detected. In fact, if an odd number of bits (not just one) have been altered, the parity will not match, but if an even number of bits have been reversed, the parity will match even though an error has occurred. However, a statistical analysis of data transmission errors has shown that a single-bit error is much more probable than a multiple-bit error in the presence of random noise, so parity is a reliable method of error detection.

Another approach to error detection involves the computation of a checksum. In this case, the packets that constitute a message are added arithmetically, and checksum number is appended to the packet sequence, calculated such that the sum of data digits plus the checksum is zero. When received, the packet sequence may be added, along with the checksum, by a local microprocessor. If the sum is not zero, an error has occurred. As long as the sum is zero, it is highly unlikely (but not impossible) that any data have been corrupted during transmission.

Data compression (and decompression) is often used to minimize the amount of data to be transmitted or stored. As with any transmission, compressed data transmission only works when both the sender and receiver of the message understand and execute the same encoding (and decoding) scheme. Two algorithms, lossless and lossy, are used for compression and decompression. Many encoding (and decoding) schemes, such as the well-known Huffman and Manchester methods, have been developed in the past decades.

Huffman coding is an entropy encoding and decoding algorithm used for lossless data compression and decompression. The term refers to the use of a variable-length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol.

Manchester coding is a line code in which the encoding of each data bit has at least one transition and occupies the same time. Manchester coding offers more complex codes, e.g. 8B/10B encoding, which use less bandwidth to achieve the same data rate (but which may be less tolerant of frequency errors and jitter in the transmitter and receiver reference clocks).

Compression is useful because it helps reduce the consumption of expensive resources, such as hard-disk space or transmission bandwidth. Large volumes of data need to be stored nowadays and massive amounts are carried over transmission links. Compressing data reduces storage and transmission costs, but compressed data must be decompressed to be used, which may be detrimental to some applications.

The encoding and decoding algorithms developed for data compression have been extended to data communication security. This topic, data communication security, is beyond the coverage of this book. Readers who are interested in it please refer to the relevant literature for further studies.

14.1.1 Data transmission over different distances

Data are typically grouped into packets that are 8, 16, 32, 64, 128 or 256 bits long, and passed between temporary holding units called registers that can be parts of a memory or an input/output (I/O) port of an integrated circuit (IC) chip or chipset. Data within a register are available in parallel because each bit exits the register on a separate conductor. To transfer data from one register to another, the output conductors of one register are switched onto a channel of parallel wires referred to as a bus. The input conductors of another register, which is also connected to the bus, capture the transferred data. Following a data transaction, the content of the source register is reproduced in the destination register. It is important to note that after any digital data transfer, the source and destination registers are equal; the source register is not erased when the data are sent. Bus signals that exit central processing unit (CPU) chips and other circuitry are electrically capable of traversing about one foot of a conductor on an integrated circuit board, or less if many devices are connected to it. Special buffer circuits may be added to boost the bus signals enough for transmission over several additional centimeters of conductor length, or for distribution to many other chips.

Both transmit and receive switches, as shown in Figure 14.3, are electronic, and operate in response to commands from a CPU. It is possible that two or more destination registers will be switched on to receive data from a single source, but only one source may transmit data onto the bus at any given time. If multiple sources were to attempt transmission simultaneously, an electrical conflict would occur when bits of opposite value are driven onto a single bus conductor. Such a condition is referred to as a bus contention. Not only will a bus contention result in the loss of information, but it also may damage the electronic circuitry. As long as all registers in a system are linked to one central control unit, bus contentions should never occur if the circuit has been designed properly.

When the source and destination registers are part of an integrated circuit (within a CPU chip, for example), they are extremely close (within some thousandths of an inch). Consequently, the bus signals are at very low power levels, may traverse a distance in very little time, and are not very susceptible to external noise and distortion. This is the ideal environment for digital transmissions. However, it is not yet possible to integrate all the necessary circuitry for a computer (i.e., CPU, memory, peripherals such as disk control, video and display drivers, etc.) on a single chip. When data are sent off-chip to another integrated circuit, the bus signals must be amplified by the conductors extended out of the chip as external pins. Amplifiers may be added to the source register, as shown in Figure 14.3.

Data transferring between a CPU and its peripheral chips generally need mechanisms that cannot be situated within the chip itself. A simple technique to tackle this might be by extending the internal

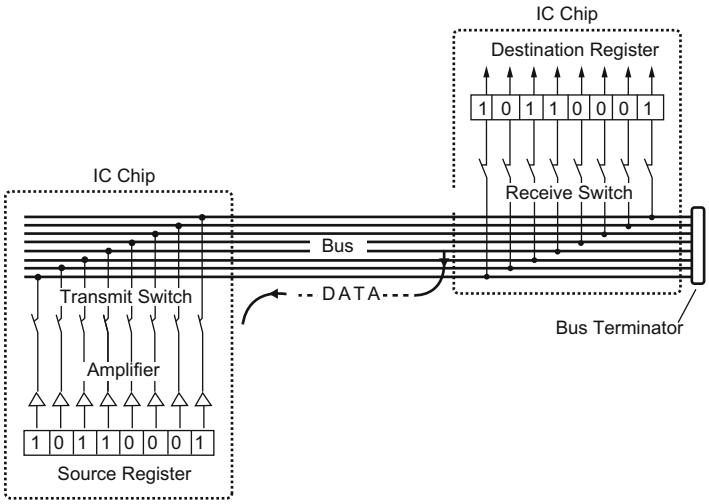


FIGURE 14.3

The data transfer mechanism within an integrated-circuit board.

buses with a cable to reach the peripheral. However, this would expose all bus transactions to external noise and distortion. When transmission with the peripheral is necessary, data are first deposited in the holding register by the CPU. These data will then be reformatted, sent with error-detecting codes, and transmitted at a relatively slow rate by digital hardware in the bus interface circuit. Data sent in this manner may be transmitted in byte-serial format if the cable has eight parallel channels, or in bit-serial format if only a single channel is available.

When relatively long distances are involved in reaching a peripheral device, driver circuits are required after the bus interface unit to compensate for the electrical effects of long cables.

However, if many peripherals are connected, or if other IC chips are to be linked, a local area network (LAN) is required, and it becomes necessary to drastically change both the electrical drivers and the protocol to send messages through the cable. Because multiple cables are expensive, bit-serial transmission is almost always used when the distance exceeds about 10 meters.

In either a simple extension cable or a LAN, a balanced electrical system is used for transmitting digital data through the channel. The basic idea is that a digital signal is sent on two wires simultaneously, one wire expressing a positive voltage image of the signal and the other a negative one. Figure 14.4 illustrates the working principle. When both wires reach the destination, the signals are subtracted by a summing amplifier, producing a signal swing of twice the value found on either incoming line. If the cable is exposed to radiated electrical noise, a small voltage of the same polarity is added to both wires in the cable. When the signals are subtracted, the noise cancels and the signal emerges from the cable without noise.

A great deal of technology has been developed for LAN systems to minimize the amount of cable required and maximize the throughput. Costs have been concentrated in the electrical-interface card

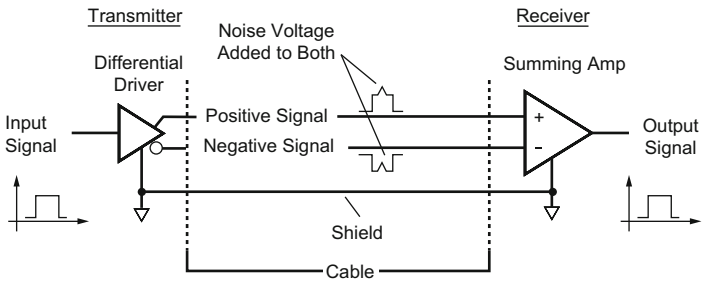


FIGURE 14.4

A balance circuit used for transmitting digital data over a long distance.

that is to drive the cable, and in the transmission software, not in the cable itself. Thus, the cost and complexity of a LAN are not particularly affected by the distance between stations.

14.1.2 Electric and electromagnetic signal transmission modes

Most modes of transmitting electrical and electromagnetic signals show up in most data transmission devices. Full-duplex, for example, might not give the best performance when most data are moving in one direction; often the case for data transmission, so some full-duplex equipment has the option of being operated in a half-duplex mode for optimal transmission.

(1) Simplex transmission mode

Simplex transmission is a mode in which data only flow in one direction, as illustrated at the top of Figure 14.5. It is used for certain point-of-sends terminals in which send data are entered without needing a corresponding reply. Radio or TV is an example of simplex transmission.

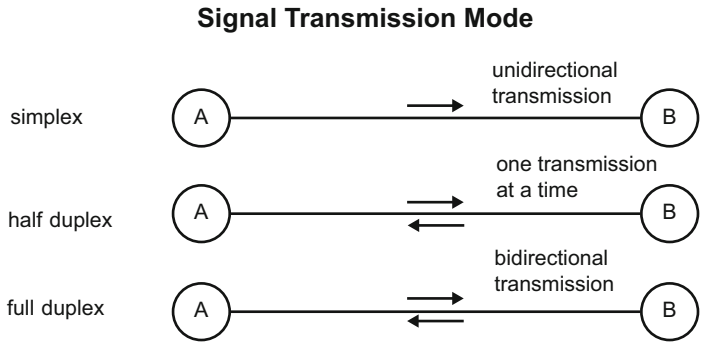


FIGURE 14.5

Three electrical signal transmission modes.

(2) Half-duplex transmission mode

Half-duplex transmission is a two-way flow of data between computer terminals. Data travel in two directions, but not simultaneously only one direction can be active at a time. This mode is commonly used for linking computers together over telephone lines, as is illustrated in the middle section of Figure 14.5.

(3) Full-duplex transmission mode

The fastest directional mode of transmission is full-duplex transmission. Here, data are transmitted in both directions simultaneously on the same channel, so it can be thought of as similar to automobile traffic on a two-lane road. It is made possible by devices called multiplexers, and is primarily limited to mainframe computers because of the expensive hardware required. This case is illustrated in the bottom section of Figure 14.5.

(4) Multiplexing transmission modes

Multiplexing is sending multiple signals or streams of information on a carrier at the same time in the form of a single, complex signal, and then recovering the separate signals at the receiving end. In analog transmission, signals are commonly multiplexed using frequency-division multiplexing (FDM), in which the carrier bandwidth is divided into subchannels of different frequency widths, each carrying a signal at the same time in parallel. In digital transmission, signals are commonly multiplexed using time-division multiplexing (TDM), in which the multiple signals are carried over the same channel in alternating time slots. In some optical-fiber networks, multiple signals are carried together as separate wavelengths of light in a multiplexed signal using dense wavelength division multiplexing (DWDM).

Frequency-division multiplexing (FDM) is a scheme in which numerous signals are combined for transmission on a single transmission line or channel. Each signal is assigned a different frequency (subchannel) within the main channel. When FDM is used in a transmission network, each input signal is sent and received at maximum speed at all times, but if many signals must be sent along a single long-distance line, the necessary bandwidth is large, and careful engineering is required to ensure that the system will perform properly.

Time-division multiplexing (TDM) is a method of putting multiple data streams in a single signal by separating the signal into many segments, each having a very short duration. Each individual data stream is reassembled at the receiving end based on timing.

Dense wavelength division multiplexing (DWDM) is a technology that puts data from different sources together on an optical fiber, with each signal being carried at the same time, at its own separate wavelength. Using DWDM, up to 80 (and theoretically more) separate wavelengths or channels of data can be multiplexed into a light stream transmitted on a single optical fiber. Each channel carries a time division multiplexed (TDM) signal. In a system with each channel carrying 2.5 Gbps (billion bits per second), up to 200 billion bits can be delivered a second by the optical fiber. DWDM is also sometimes called wave division multiplexing (WDM).

Since each channel is demultiplexed at the end of the transmission to retrieve the original source, different data formats can be transmitted together. Specifically, Internet Protocol (IP) data,

Synchronous Optical Network data (SONET), and asynchronous transfer mode (ATM) data can all travel at the same time within the optical fiber. DWDM promises to solve the "fiber exhaust" problem and is expected to be the central technology in the all-optical networks of the future.

14.2 DATA TRANSMISSION I/O DEVICES

Within a microprocessor chip, digital signals travel from one component to another through an integrated circuit termed a data bus. One part of the bus runs between the prime (system or main) memory and microprocessor, and another connects the prime memory to various storage and peripheral devices. The segment of the data bus that extends between the prime memory and peripheral devices is called the extension bus, or I/O bus in most cases. As signals move along the I/O bus, they can travel through expansion slots, cards, ports and connectors to peripheral devices.

An I/O port can only host one client; in contrast, an I/O bus can serve several. This section will introduce these I/O buses, including PCI (peripheral component interconnect), ISA (industry standard architecture), USB (universal serial bus), Firewire or IEEE-1394, and IEEE-488. The transmission ports discussed include AGP (accelerated graphics port), parallel ports, IDE (integrated drive electronics) ports, and SCSI (small computer system interface) ports.

The RS (recommended standards) series mainly include RS-232, RS-422, RS-485, and RS-530. They are used as the connectors connecting DTEs such as computers and programmable controllers with DCEs such as modems. The RS series are standard interfaces between a computer and some special peripherals such as modems, mouse, and keyboard, etc.

14.2.1 I/O buses

(1) PCI (peripheral component interconnect) bus

Intel has developed a standard interface, named the PCI local bus, for microprocessor chips. This technology allows fast memory, disk and video access. It is now the main interface bus used in most programmable controllers, and is rapidly replacing the ISA bus for internal interface devices. It is very adaptable, and most external buses, such as SCSI and USB, connect to the microprocessor by using it.

The PCI local bus has two variants: conventional PCI and the PCI-X bus. PCI-X is a high-performance variant of 64-bit PCI designed for servers. PCI-X adapters and slots are backward-compatible with 32-bit PCI slots and adapters.

The PCI bus transfers data using the system clock and can operate over a 32-bit or 64-bit data path. If data are transferred at 64 bits at a rate of 33 MHz, then the maximum transfer rate is 264 Mbps. The 32-bit or 64-bit memory address space can reach 4 gigabytes or 16 exabytes, respectively. Each device slot of a PCI local bus can have a configuration space up to 256 bytes. The PCI-X specification also provides options for 3.3 V signalling, 64-bit bus width, and 66 MHz clocking on server motherboards.

The PCI operations, including the bus transaction, bus arbitration, bus configuration, and PCI bus interrupt handling are given briefly in the following.

(1) PCI bus transactions

Each PCI bus transaction is made up of an address phase followed by one or more data phases. Any PCI bus device (or client) may initiate a bus transaction. First, this device must request permission from a PCI bus arbiter on the microprocessor chip. The arbiter grants permission to one of the requesting devices, which then becomes the bus master or initiator. The bus master begins the address phase by broadcasting a 32-bit address plus a 4-bit command code, and then waits for a target to respond. All other devices examine this address and one of them responds a few cycles later.

The direction of the data phases may be from initiator to target (write transaction) or vice versa (read transaction), but all of the data phases must be in the same direction. Either party may pause or halt the data phases at any point. While the PCI bus transfers 32 bits per data phase, the initiator transmits a 4-bit byte mask indicating which 8-bit bytes are to be considered significant. In particular, a masked write must affect only the desired bytes in the target PCI device.

The commands that refer to cache lines depend on the PCI configuration space cache line size register being set up properly; they may not be used until that has been done. There are 16 possible 4-bit command codes, and 12 of them are assigned. With the exception of the unique dual address cycle, the least significant bit of the command code indicates whether the following data phases are a read (data sent from target to initiator) or a write (data sent from an initiator to target). PCI targets must examine the command code as well as the address and not respond to address phases which specify an unsupported command code.

(2) PCI bus arbitrations

The PCI bus arbiter is used to efficiently manage the accesses to a PCI bus that is shared by several masters. Access to the PCI bus is automatically determined by the individual priorities of each master. At any given time, more than one PCI bus initiator may request use of the PCI bus by asserting its specific request signal (REQn). The arbiter determines which PCI bus initiator controls the PCI bus, by asserting that device's specific grant signal (GNTn). Figure 14.6(a) shows the PCI arbiter interface signals and Figure 14.6(b) illustrates the relationship of the PCI bus initiator device with the bus arbiter.

The arbiter can use two algorithms to resolve any conflicts due to multiple requests. The first is "pure rotation", which is a turn-based method that allows each bus master one transaction in turn. If only one master requests the bus, that master will immediately get the grant. Figure 14.7(a) illustrates the pure rotation scheme.

"Fair rotation" is the second algorithm which is employed if an embedded microprocessor is required to initialize the system, but will not be used after that point. By giving it highest priority, the fair rotation scheme allows the embedded processor access to the PCI bus on every other transaction when it is requesting the bus continuously. The other masters use a pure rotation scheme. The fair rotation scheme is illustrated in Figure 14.7(b).

(3) PCI bus configurations

As mentioned, each PCI bus device can have a block of 256 bytes of configuration space: 16 doublewords (each double word consists of 32 bits) are taken as header information plus 48 doublewords of device-specific configuration registers. The header contains a vendor ID and device type, flags which show whether the device generates interrupts, whether the device is 66 MHz-capable and other low-level performance-related information. It also contains the base address locations of I/O ports, RAM and expansion ROM, the maximum latency register (mentioned in subsection 5.1.1) and

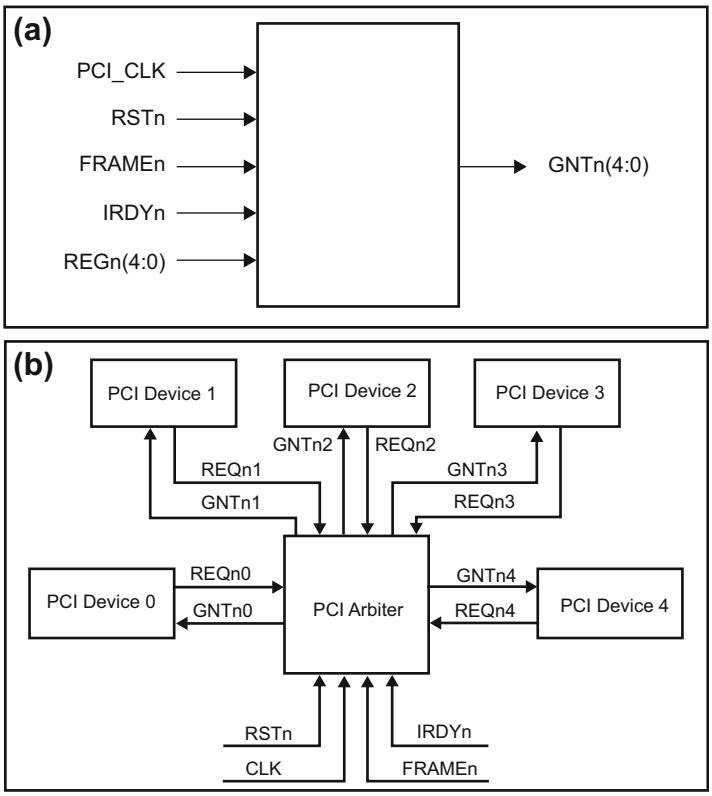


FIGURE 14.6

PCI bus arbiter core: (a) PCI arbiter core interface signals; (b) the interface of the PCI bus initiators with the arbiter.

other similar general information. ROMs can contain code for different microprocessor architectures, and a configuration register shows which ones are supported.

In a typical system, the firmware (nucleus or kernel of the operating system) queries all PCI buses at start-up time via the PCI configuration space to find out what devices are present and what system resources (such as memory space, I/O space, interrupt lines, etc.) each needs. It then allocates the resources and tells each device what its allocation is. The PCI configuration space also contains a small amount of device type information, which helps an operating system to choose device drivers for it, or at least to have a dialogue with a user about the system configuration. These are typically necessary for devices used during system start-up, before device drivers are loaded by the operating system.

(4) PCI bus interrupts

The PCI bus includes four interrupt lines (INTA#, INTB#, INTC#, and INTD#), all of which are available to each device. However, they are not wired in parallel like the other PCI bus lines.

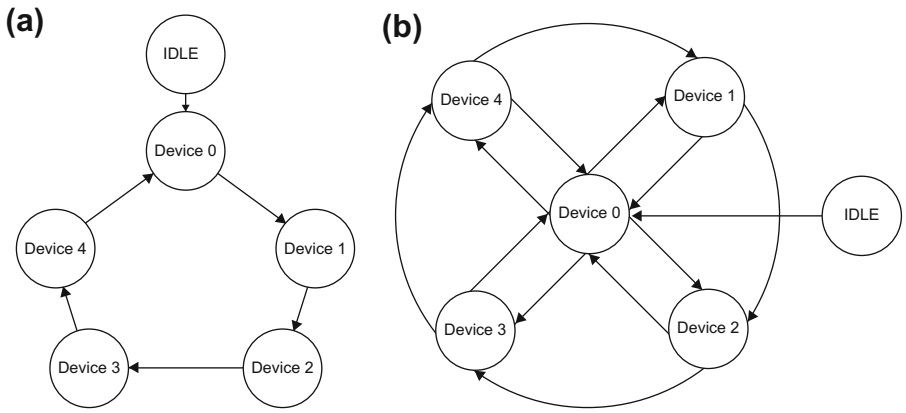


FIGURE 14.7

PCI bus arbitration algorithms: (a) pure rotation arbitration; (b) fair rotation arbitration.

The positions of the interrupt lines rotate between slots, so what appears to one device as the INTA# line is INTB# to the next and INTC# to the one after that. Single-function devices use their INTA# for interrupt signaling, so the device load is spread fairly evenly across the four available interrupt lines. This alleviates a common problem with sharing interrupts.

PCI bridges (between two PCI buses) map the four interrupt traces on each of their sides in varying ways. Some use a fixed mapping, and in others it is configurable. Software generally cannot determine which interrupt line a device's INTA# pin is connected to across a bridge. The mapping of PCI interrupt lines onto system interrupt lines, through the PCI host bridge, is similarly implementation-dependent. The result is that it can be impossible to determine how a PCI device's interrupts will appear to software. Platform-specific BIOS (basic input/output system) code is meant to know this, and to set a field in each device's configuration space indicating which the interrupt request signal, IRQ, is connected to, but this process is not reliable.

Later revisions of the PCI specification add support for message-signalled interrupts. In this system a device signals its need for service by performing a memory write, rather than by asserting a dedicated line. This alleviates the problem of scarcity of interrupt lines, and even if interrupt vectors are still shared, it does not suffer the sharing problems of level-triggered interrupts. It also resolves the routing problem, because the memory write is not unpredictably modified between device and host. Finally, because the message signaling is in-band, it resolves some synchronization problems that can occur with posted writes and out-of-band interrupt lines.

(2) ISA (industry standard architecture) bus

IBM developed the Industry Standard Architecture or ISA bus for their 80286-based AT (advanced technology) computer (IBM AT). Apart from specialized industrial use, ISA bus is all but gone today. Even where present, system manufacturers often shield customers from the term "ISA bus", referring to it instead as the "legacy bus". In practice there is no speed difference between running many serial communication peripherals using a PCI or an ISA bus, though the PCI advantage is obvious for

high-speed devices such as video cards. Thus, there is no reason to convert an ISA serial communication system to PCI bus, as ISA will provide equivalent functionality, generally at a lower price.

Initially the ISA bus had the advantage of being able to deal with 16 bits of data at a time. An extra edge connector was added to give compatibility with a PC/AT bus. This gives an extra eight additional data lines for a total of 16 bits, and four address lines for a total of 24 bits. Thus, the ISA bus has a 32-bit data and a 24-bit address bus. It also adds new interrupt lines connected to a second 8259 PIC (programmable interrupt controller) connected to one of the lines of the first and four 16-bit DMA (direct memory access) channels, as well as control lines to select 8- or 16-bit transfers.

The PC/AT bus was designed with an expansion bus which not only took advantage of the new technology, but also remained compatible with the older-style 8-bit PC/XT add-in boards. Anticipating that advances in microprocessors would again outpace advances in bus technology, the PC/AT bus had two separate oscillators. In this way, the microprocessor and expansion bus could be run on different clocks with different speeds. Therefore, a controller or computer running a newer processor with 33 MHz clock speed could also run its expansion bus at an 8 MHz clock rate. ISA cards are more cumbersome to install than other cards because I/O addresses, interrupts, and clock speed must be set using jumpers and switches on the card itself. Other bus options, which use software to set these parameters, are called plug-and-play. While there is nothing inferior about using jumpers and switches, it can be more intimidating for novice users. The ISA system, however, does not have a central registry from which to allocate system resources. Consequently, each device behaves as though it has sole access to system resources such as DMA, I/O ports, IRQs, and memory. Obviously, this can cause problems when using multiple add-in boards in a single system.

Figure 14.8 shows a typical connection to the ISA bus. The ALE (sometimes known as BALE) controls the address latch; when active low, it latches the address lines A2 A19 to the ISA bus. The address is latched when ALE goes from a high to a low. The Pentium's data bus is 64 bits wide,

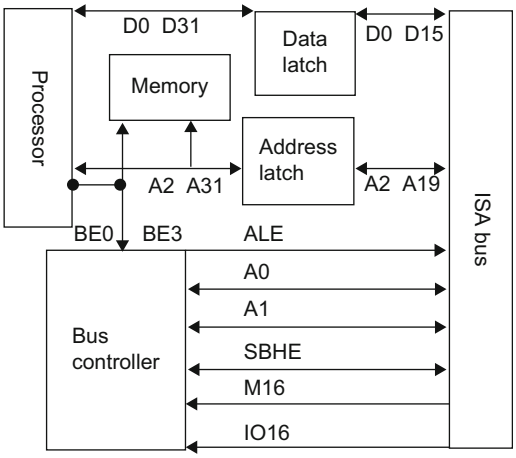


FIGURE 14.8

One configuration of the ISA bus connections.

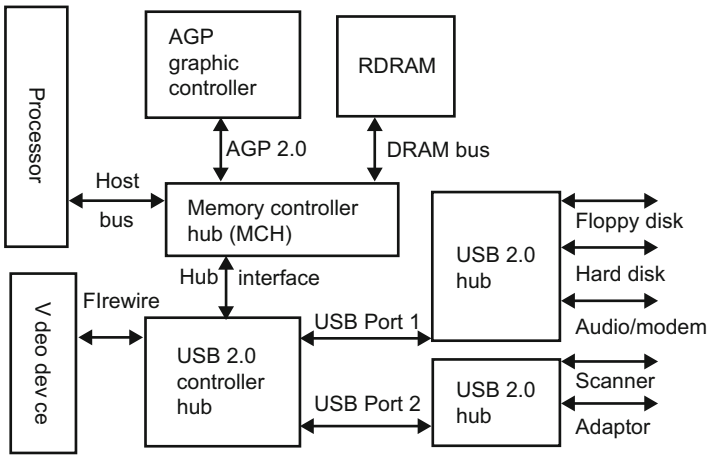


FIGURE 14.9

An example connection of the USB 2.0 system.

whereas the ISA expansion bus is 16 bits wide. It is the bus controller's function to steer data between the microprocessor and the slave device for either 8-bit or 16-bit communications.

(3) USB (universal serial bus) bus

The USB (universal serial bus) is mainly used for communication between a computer and peripheral devices, replacing many varieties of serial and parallel ports for the connection of medium-bandwidth peripherals such as keyboards, scanner, modem, video, game or graphic controller, music interface, etc. The great advantage of USB is that it allows for peripherals to be added and deleted from the system without causing any upsets. The system will also automatically sense the connected device and load the required driver.

USB is a balanced bus architecture that hides the complexity of the operation from the devices connected to it. The USB host controller controls system bandwidth. Each device is assigned a default address when first powered or reset, and hubs and functions are assigned a unique device address. All USB devices are attached to the USB via a port. Hubs indicate the attachment or removal of a USB device in its port status. Figure 14.9 shows an example of the USB 2.0 system, where a memory hub is used to provide a fast data transfer, while the Firewire connection provides ultrahigh-speed connection for video transfers. The USB connection provides low-high and full-speed connections to most of the peripherals that connect to the system. Such connections can be internal or can connect to an external hub. There are two main methods to implement USB, as given below.

1. Open host controller interface (OHCI). This method defines the register-level interface that enables the USB controller to communicate with the host computer and the operating system. OHCI is an industrial standard hardware interface for operating systems, device drivers, and the BIOS to

manage the USB. It optimizes performance of the USB bus while minimizing CPU overhead by using scatter/gather bushmaster hardware support. It has efficient isochronous data transfers allowing for high USB bandwidth without slowing down the host CPU, and ensures full compatibility with all USB devices.

2. Universal host controller interface (UHCI). This method defines how the USB controller talks to the host computer and its operating system. It is optimized to minimize host computer design complexity and uses the host CPU to control the USB bus. This method has the advantage of simple design, which reduces the transistor count required to implement it, thus reducing the system cost. Furthermore, it can provide full compatibility with all USB devices.

In data transmission, UHCI supports two types of transfers: stream and message. A stream has no defined structure, whereas a message does. At start-up, one message pipe, control pipe 0, always exists as it provides access to the device's configuration, status, and control information. USB optimizes large data transfers and real-time data transfers. When a pipe is established for an endpoint, most of its transfer characteristics are determined, and remain fixed for its lifetime. Each bus transaction involves the transmission of up to three packets, which can be (1) token packet transmission, (2) data packet transmission, and (3) handshake packet transmission. With these transfer characteristics, USB defines the following four transfer types:

(i) Control transfers. This is bursty, aperiodic, host-software-initiated request/response communication typically used for command/status operations.
(ii) Isochronous transfers. This is periodic, continuous communication between host and device typically used for time-relevant information. This transfer type also preserves the concept of time encapsulated in the data. This does not imply, however, that the delivery needs of such data are always time-critical.
(iii) Interrupt transfers. This is small data, aperiodic, low-frequency, bounded latency, device-initiated communication typically used to notify the host of device service needs.
(iv) Bulk transfers. Aperiodic, large, bursty communication typically used for data that can use any available bandwidth and also is delayed until bandwidth is available.

As mentioned earlier, a major advantage of USB is the hot attachment and detachment of devices. USB does this by sensing when a device is attached or detached. When this happens, the host system is notified, and system software interrogates the device, determines its capabilities, and automatically configures it. All the required drivers are then loaded and applications can immediately make use of the connected device.

(a) Attachment of USB devices

All USB devices are addressed using the USB default address when initially connected or after they have been reset. The host determines whether the device is a hub or a function and assigns it a unique USB address. The host then establishes a control pipe using assigned USB address and endpoint number zero. If the device is a hub and USB devices are attached to its ports, then the above procedure is followed for each of the attached devices. If the attached USB device is a function, then attachment notifications will be dispatched by USB software to interested host software.

(b) Removal of USB devices

When a USB device has been removed from one of its ports, the hub automatically disables the port and provides an indication of device removal to the host. If the removed USB device is a hub, then the removal process must be performed for all of the USB devices previously attached to it. If the removed USB device is a function, removal notifications are sent to interested host software.

(4) Firewire or IEEE-1394 Bus

The main competitor to USB is the Firewire standard (IEEE 1394 1995 buses), which is a high-speed serial bus typically used for video transfers, whereas USB supports a low-to-medium-speed serial bus for a greater range of peripherals. Firewire supports rates of approximately 100, 200, and 400 Mbps, known as S100, S200, and S400, respectively. The future standard promises higher data rates, and ultimately it is envisaged that rates of 3.2 Gbps will be achieved, when optical fiber is introduced into the system. It uses point-to-point interconnect with a tree topology: 1000 buses with 64 nodes gives 64,000 nodes. Firewire can also have automatic configuration and hot plugging, and can perform both asynchronous and isochronous data transfer, where a fixed bandwidth is dedicated to a particular peripheral. This should subsequently reduce the costs of production of controller interfaces and peripheral connectors, as well as simplifying the requirements placed on users when setting up their devices. Firewire is therefore a more economical interface bus standard that performs fast and high-bandwidth data transmissions but it does have a maximum cable length of 4.5 meters.

There are two bus categories in Firewire:

(a) Cable. This is a bus that connects external devices via a cable. This cable environment is an acyclic network with finite branches consisting of bus bridges and nodes (cable devices). Acyclic networks contain no loops and result in a tree topology, with devices daisy-chained and branched (where more than one device branch is connected to a device). Devices on the bus are identified by node IDs. Configuration of the node IDs is performed by the self ID and tree ID processes after every bus reset. This happens every time a device is added to or removed from the bus, and is invisible to the user.

(b) Backplane. This type of topology is an internal bus. An internal IEEE-1394 device can be used alone, or incorporated into another backplane bus. Implementation of the backplane specification lags behind the development of the cable environment, but one could imagine internal IEEE-1394 hard disks in one computer being directly accessed by another IEEE-1394 connected computer.

One of the key capabilities of IEEE-1394 is isochronous data transfer, although both asynchronous and isochronous are supported, and are useful for different applications. Isochronous transmission transfers data such as real-time speech and video, both of which must be delivered uninterrupted, and at the rate expected, whereas asynchronous transmission is used to transfer data that are not tied to a specific transfer time. With IEEE-1394, asynchronous transmission is generally used to send data to an explicit address, and get confirmation when it is received. Isochronous transfer, however, is an unacknowledged guaranteed bandwidth transmission method, useful for just-in-time delivery of multimedia-type data.

An isochronous "talker" requests an amount of bandwidth and a channel number. Once the bandwidth has been allocated, it can transmit data preceded by a channel ID. The isochronous listeners can then listen for the specified channel ID and accept the data following it. If the data are not intended for a node, it will not be set to listen on the specific channel ID. Up to 64 isochronous channels are available, and these must be allocated, along with their respective bandwidths, by an isochronous resource manager on the bus.

By comparison, asynchronous transfers are sent to explicit addresses on the IEEE-1394 bus. When data are to be sent, they are preceded by a destination address, which each node checks, to identify packets for itself. If a node finds a packet addressed to itself, it copies it into its receive buffer. Each node is identified by a 16-bit ID, containing the 10-bit bus ID and 6-bit node or physical ID. The actual packet addressing, however, is 64 bits wide, providing a further 48 bits for addressing a specific offset within a node's memory.

(5) IEEE-488 bus

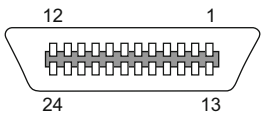
The IEEE 488 is a common parallel interface standard, originally developed by the Hewlett-Packard Corporation in 1974, and called the HP-IB (Hewlett-Packard interface bus). This interface bus became so popular that it was adopted by the IEEE (Institute of Electrical and Electronics Engineers) in 1975 and designated the IEEE-488 standard. In common practice, IEEE-488 is referred to as the GPIB interface (general purpose interface bus), so IEEE-488, GPIB, and HP-IB all refer to the same parallel interface standard.

The primary purpose of the GPIB is to interface laboratory instruments with each other, and with computers. Both devices must have a GPIB interface card installed, and the computer must be running a GPIB controller program for the connection to operate.

Almost any instrument can be used with the IEEE-488 specification, because it is indepentdent of the function of the instrument itself, or about the form of the instrument's data. Instead, the specification defines a separate component, the interface, which can be added to the instrument. The signals passing into the interface from the IEEE-488 bus and from the instrument are defined in the standard. The instrument does not have complete control over the interface. Often the bus controller tells the interface what to do. The active controller performs the bus control functions for all the bus instruments.

There are 24 lines in the GPIB interface bus and cable (Figure 4.10). Eight of these lines carry data, five are for device control, three are handshaking lines, and eight are signal grounds. IEEE-488 standards give two main types: IEEE-488.1 and IEEE-488.2.

1. The IEEE-488.1 standard greatly simplifies the interconnection of programmable instruments by clearly defining mechanical, hardware, and electrical protocol specifications. For the first time, instruments from different manufacturers were connected by a standard cable. This standard does not address data formats, status reporting, message exchange protocol, common configuration commands, or device-specific commands.
2. The IEEE-488.2 standard enhances and strengthens IEEE-488.1 by specifying data formats, status reporting, error handling, controller functionality, and common instrument commands. It focuses mainly on software protocol issues and thus maintains compatibility with the hardware-oriented IEEE- 488.1 standard. IEEE-488.2 systems tend to be more compatible and reliable.



Pin	Name	Description	Source
1	DIO1	Data Bit 1	Talker
2	DIO2	Data Bit 2	Talker
3	DIO3	Data Bit 3	Talker
4	DIO4	Data Bit 4	Talker
5	EOI	End Or Indentity	Talker/Controller
6	DAV	Data Valid	Controller
7	NRFD	Not Ready For Data	Listener
8	NDAC	No Data Accepted	Listener
9	IFC	Interface Clear	Controller
10	SRQ	Service Request	Talker
11	ATN	Attention	Controller
12		Shield	
13	DIO5	Data Bit 5	Talker
14	DIO6	Data Bit 6	Talker
15	DIO7	Data Bit 7	Talker
16	DIO8	Data Bit 8	Talker
17	REN	Remote Enabled	Controller
18		Ground DAV	
19		Ground NRFD	
20		Ground NDAC	
21		Ground IFC	
22		Ground SRQ	
23		Ground ATN	
24		Logical Ground	

FIGURE 14.10

The 24 pinout of the IEEE-488 interface bus.

14.2.2 I/O ports

(1) AGP (accelerated graphics port) port

As computers became increasingly graphically oriented, successive generations of graphics adapters began to push the limits of PCI bus bandwidth. This led to the development of new bus architectures dedicated to graphics adapters. The accelerated graphic port (AGP) is a major advance in the connection of three-dimensional graphics applications, and is based on an enhancement of the PCI bus.

The primary advantage of AGP over PCI is that it provides a dedicated pathway between the slot and the microprocessor, rather than sharing the PCI bus, resulting in faster transfer between the main system memory and the local graphic card, so reducing the need for large areas of memory on the graphics card.

To understand the advantages of AGP graphic implementation, it is necessary to understand the issues that AGP technology has resolved. Figure 14.11(a) shows an architectural diagram of a generic PCI bus-based graphics subsystem. In this architecture, the graphics subsystem resides on the PCI bus. Note that the PCI bus graphics adapter embeds its own local memory on the adapter card. This architecture raises several issues that motivated the need for AGP. Since graphics data such as textures

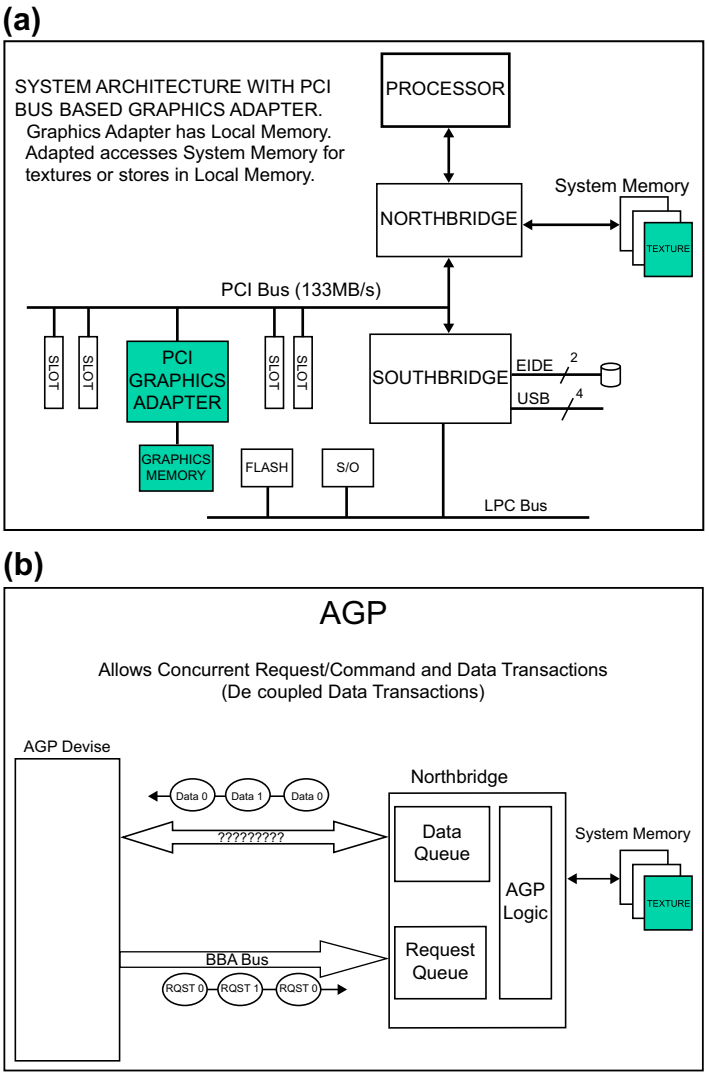


FIGURE 14.11

PCI-based and AGP-based graphic implementations in microprocessor chipsets: (a) a bus architecture of PCI-based graphic implementation; (b) a bus architecture of AGP-based graphic implementation.

are stored in main system memory, the PCI bus-based graphics card must access this via the PCI bus. These accesses can occur frequently, particularly if the graphics adapter has a small amount of local memory. Unfortunately, the graphics card must compete with other PCI bus peripherals for bandwidth which may starve other peripherals if these accesses are frequent.

Figures 14.11(b) illustrates how AGP technology elegantly resolves these issues. The system controller embeds the AGP graphic interface, which then uses the PCI bus protocol in tandem with a sideband addressing (SBA) bus for concurrent posting of commands from the graphics card. The Northbridge embeds read and write and command queues (buffers) to allow full-speed data and command transport between the AGP device and the system controller, and concurrent full-speed data transport between the system controller and the system memory.

In summary, the native architecture of the AGP graphics subsystem offers significant raw performance improvement over PCI bus-based graphics subsystems, allowing the graphics subsystem to view and use main memory just like its own local memory meaning the AGP graphics card shares system memory. The card cannot distinguish between system and local memory; it all appears to be local. To the end-user, graphics performance can be enhanced by increasing system memory, rather than by adding expensive graphics memory.

The graphics subsystem no longer needs to compete for PCI bus bandwidth in order to access data in system memory, which allows it to run at full speed with minimal interruption from other components in the system. It also increases system concurrency meaning that the processor, AGP graphics subsystem, and PCI bus device can run independently and concurrently, thus increasing system performance.

There are several modes (data transfer rates) that have evolved over time. The first version is the original AGP-1X mode, offering a data transfer rate of up to 264 Mbps. Second is AGP-2X mode, doubling the data transfer rate to up to 528 Mbps. The third is AGP-4X, offering a transfer rate of up to 1 Gbps. Finally, the fourth version, AGP-8X, offers the highest performance data transfer rate of up to 2.1 Gbps.

(2) Parallel ports

A parallel port is a type of interface found on computers (personal and otherwise) for connecting various peripherals. The IEEE-1284 standard defines the bidirectional version of the parallel port. In its standard form, it allows only for simple communications from the PC outwards. However, like the RS-232, the parallel port is a standard port of the PC.

All parallel ports use a bidirectional link in either a compatible, nibble, or byte mode. These modes are relatively slow, as the software must monitor the handshaking lines (up to 100 kbps). To allow high speeds, the enhanced parallel port and extended capabilities port protocol modes allow high-speed data transfer using automatic hardware handshaking.

(3) IDE (integrated drive electronics) ports

The most popular interface for hard disk drives is the integrated drive electronics (IDE) interface. Its main advantage is that the controller is built into the disk drive, and the interface to the motherboard consists simply of a stripped-down version of the ISA (industry standard architecture) bus. The most common standard is the ANSI-defined ATA-IDE standard. It uses a 40-way ribbon cable to connect to 40-pin header connectors. The standard allows for the connection of two disk drives in a daisy-chain

configuration, which can cause problems because both drives have controllers within them. The primary drive is assigned as the master, and the secondary driver is the slave, set by setting jumpers on the disk drive. They can also be set by software that uses the cable select pin on the interface. The specifications for the IDE include: (1) maximum of two devices (hard disks); (2) maximum capacity for each disk of 528 MB; (3) maximum cable length of 18 inches; (4) data transfer rates of 3.3, 5.2, and 8.3 Mbps.

A new standard called enhanced IDE (E-IDE) allows for higher capacities: (1) maximum of four devices (hard disks); (2) uses two ports (for master and slaves); (3) maximum capacity for each disk of 8.4 GB; (4) maximum cable length of 18 inches; (5) data transfer rates of 3.3, 5.2, 8.3, 11.1, and 16.6 Mbps.

The PC (personal computer) is now a highly integrated system containing a microprocessor, an IDE systems controller and a PCI IDE/ISA accelerator, as illustrated in Figure 14.12. The IDE system controller provides the main interface between the processor and the level-2 cache, the DRAM, and the PCI bus. It is one of the most important devices in the system, since it ensures that data flow to and from the microprocessor in the correct way. The PCI bus links to interface devices, and also the PCI IDE/ISA accelerator (such as the PIIX4 device). The PCI IDE/ISA device then interfaces to other buses, such as IDE and ISA. The IDE interface has separate signals for both the primary and secondary IDE drives; these are electrically isolated, which allows drives to be swapped easily without affecting other ports.

The PCI IDE/ISA accelerator is a massively integrated device (the PIIX4 has 324 pins) and provides for an interface to other buses, such as USB and X-Bus. It also handles the interrupts from the PCI bus and the ISA bus. It thus has two integrated 82C59 interrupt controllers, which support up to

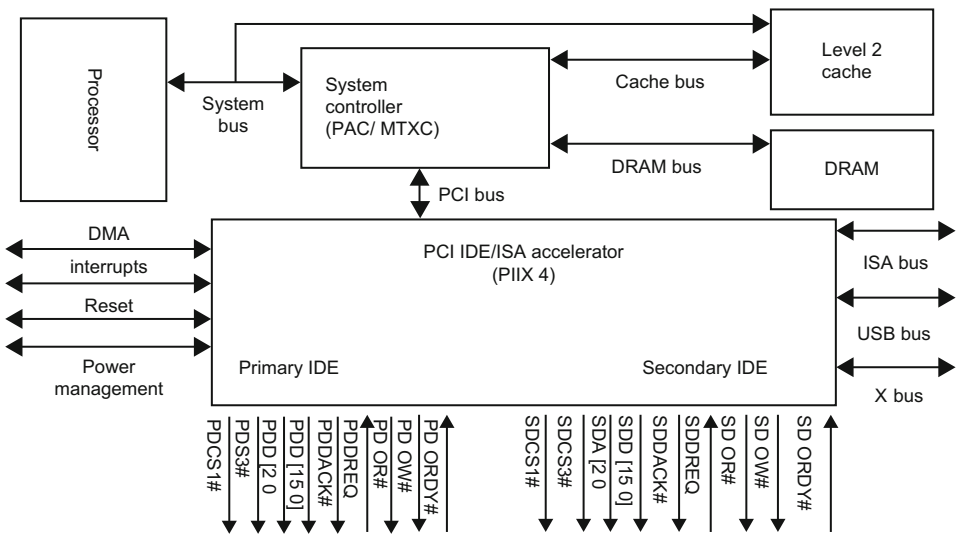


FIGURE 14.12

IDE system connections in microprocessor chipsets.

15 interrupts. The PCI IDE/ISA accelerator also handles DMA (direct memory access) transfers (on up to eight channels), and thus has two integrated 82C37 DMA controllers. Along with this, it has an integrated 82C54, which provides for the system timer, DRAM refresh signal, and the speaker tone output.

The IDE (or AT bus) is the de facto standard for most PC hard disks. It has the advantage over older-type interfaces that the controller is integrated into the disk drive, so the computer only has to pass high-level commands to the unit, and actual control can be achieved with the integrated controller. Several companies developed a standard command set for an AT attachment (ATA). Commands include: (1) read sector buffer, which reads contents of the controller's sector buffer; (2) write sector buffer, which writes data to the controller's sector buffer; (3) check for active; (4) read multiple sectors; (5) lock drive door. Control of the disk is achieved by passing a number of high-level commands through a number of I/O port registers.

(4) SCSI (small computer system interface) ports

Small computer system interface (SCSI) is a set of interface standards for physically connecting and transferring data between computers and peripheral devices, most commonly used for hard disks and tape drives, but able to connect a wide range of other devices, including scanners, and CD and DVD drives.

The SCSI standards define commands, protocols, and electrical and optical interfaces. Within a SCSI interface, there is an intelligent bus subsystem which can support multiple devices cooperating concurrently. Each device is assigned a priority. The main types of SCSI are:

1. SCSI-I. SCSI-I transfers at rate of 5Mbps with an 8-bit data bus and seven devices per controller.
2. SCSI-II. SCSI-II supports SCSI-I and has one or more of the following features:
   - **(a)** Fast SCSI, which uses a synchronous transfer to give 10 Mbps transfer rate. The initiator and target initially negotiate to see whether they can both support synchronous transfer. If they can, they then go into a synchronous transfer mode.
   - **(b)** Fast and Wide SCSI-2, which doubles the data bus width to 16 bits to give 20 Mbps transfer rate.
   - **(c)** Fifteen devices per master device.
   - **(d)** Tagged command queuing (TCQ), which greatly improves performance and is supported by Windows, NetWare, and OS-2.
   - **(e)** Multiple commands sent to each device.
   - **(f)** Commands executed in whatever sequence will maximize device performance.
3. Ultra SCSI (SCSI-III). Ultra SCSI operates either as 8-bit or 16-bit with either 20 or 40 Mbps transfer rate (Table 14.1).

A SCSI bus is made of a host adapter connected to a number of SCSI units. Although all units connect to a common bus, only two units can transfer data at a time, either from one unit to another or from one unit to the host. The great advantage of this transfer mechanism is that it does not involve the microprocessor.

Each unit is assigned a SCSI-ID address. In the case of SCSI-I, this ranges from 0 to 7 (where 7 is normally reserved for a tape drive). The host adapter takes one of the addresses; thus a maximum of seven units can connect to the bus. Most systems allow the units to take any SCSI-ID address, but older systems used to require boot drives to be connected to a specific SCSI address. When the system is

Table 14.1 SCSI Types

Indices Types	Data Bus (bits)	Transfer Rate (MB/s)	Tagged Command Queuing	Parity Checking	Maximum Devices	Pins on Cable and Connector
SCSI-I	8	5	no	option	7	50
SCSI-II Fast	8	10 (10 MHz)	yes	yes	7	50
SCSI-III Fast/ Wide	16	20 (10 MHz)	yes	yes	15	68
Ultra SCSI	16	40 (20 MHz)	yes	yes	15	68

initially booted, the host adapter sends out a start unit command to each SCSI unit. This allows each of the units to start in an orderly manner (and not overload the local power supply). The host will start with the highest-priority address (ID = 7) and finishes with the lowest address (ID = 0). Typically, the ID is set with a rotating switch selector or by three jumpers.

SCSI defines an initiator control and a target control. The initiator requests a function from a target, which then executes the function, as illustrated in Figure 14.13, where the initiator effectively takes over the bus for the time to send a command and the target executes the command and then contacts the initiator and transfers any data. The bus will then be free for other transfers. Table 14.2 gives the definitions of the main SCSI signals. Each of the control signals can be true or false, OR-tied driven or Non-OR-tied driven. In OR-tied driven mode, the driver does not drive the signal to the false state. In this case, the bias circuitry of the bus terminators pulls the signal false whenever it is released by the drivers at every SCSI device. If any driver is asserted, then the signal is true. The BSY, SEL, and RST signals are OR-tied. In the ordinary operation of the bus, the BSY and RST signals may be simultaneously driven true by several drivers. However, in Non-OR-tied driven, the signal may be actively driven false. No signals other than BSY, RST, and D(PARITY) are driven simultaneously by two or more drivers.

The SCSI bus allows any unit to talk to any other unit, or the host to talk to any unit. Thus there must be some means of arbitration where units capture the bus. The main phases that the bus goes through are as follows:

1. Free bus. In this state, there are no units that either transfer data or have control of the bus. It is identified by deactivate SEL and BSY (both will be high). Thus, any unit can capture the bus.
2. Arbitration. In this state, a unit can take control of the bus and become an initiator. To do this, it activates the BSY signal and puts its own ID address on the data bus. After a delay, it tests the data

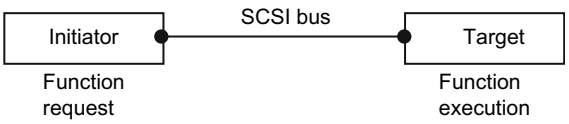


FIGURE 14.13

Initiator and target in SCSI.

Table 14.2 SCSI main signals

Signals	Definitions
BSY	Indicates that the bus is busy, or not (an OR-tied signal)
ACK	Activated by the initiator to indicate an acknowledgement for a REQ information transfer handshake
RST	When active (low) resets all the SCSI devices (an OR-tied signal)
ATN	Activated by the initiator to indicate the attention state
MSG	Activated by the target to indicate the message phase
SEL	Activated by the initiator, is used to select a particular target device (an OR-tied signal)
C/D (control/Data)	Activated by the target to identify whether there is data or control on the SCSI bus
REQ	Activated by the target to acknowledge a request for an ACK information transfer handshake
I/O (input/output)	Activated by the target to show the direction of the data on the data bus. Input defines that data are an input to the initiator, else they are an output

bus to determine whether a high-priority unit has put its own address on it. If it has, then it will allow the other unit access to the bus. If its address is still on it, then it asserts the SEL line. After a delay, it then has control of the bus.

3. Selection. In this state, the initiator selects a target unit to carry out a given function, such as reading or writing data. The initiator outputs the OR value of its SCSI-ID and the SCSI-ID of the target onto the data bus (e.g., if the initiator is 2 and the target is 5 then the OR-ed ID on the bus will be 00100100). The target then determines that its ID is on the data bus and sets the BSY line active. If this does not happen within a given time, then the initiator deactivates the SEL signal, and the bus will be free. The target determines that it is selected when the SEL signal and its SCSI-ID bit are active and the BSY and I/O signals are false. It then asserts the BSY signal within a selection abort time.
4. Reselection. When the arbitration phase is complete, the wining SCSI device asserts the BSY and SEL signals and has delayed at least a bus clear delay plus a bus settle delay. The wining SCSI device sets the DATA BUS to a value that is the logical OR of its SCSI-ID bit and the initiator's CSI-ID bit. Sometimes, the target takes some time to reply to the initiator's request. The initiator determines that it is reselected when the SEL and I/O signals and its SCSI-ID bit are true and the BSY signal is false. The reselected initiator then asserts the BSY signal within a selection abort time of its most recent detection of being reselected. An initiator does not respond to a reselection phase if other than two SCSI-ID bits are on the data bus. After the target detects that the BSY signal is true, it also asserts the BSY signal and waits a given time delay and then releases the SEL signal. The target may then change the I/O signal and the DATA BUS. After the reselected initiator detects the SEL signal is false, it releases the BSY signal. The target continues to assert the BSY signal until it gives up the SCSI bus.
5. Command. The command phase is used by the target to request command information from the initiator. The target asserts the C/D signal and negates the I/O and MSG signals during the REQ/ACK handshake(s) of this phase. The format of the command descriptor block for 6-byte commands is: Byte 0 is operation code; Byte 1 is logical unit number (MSB, if required); Byte 2

is logic block address; Byte 3 is logic block address (LSB, if required); Byte 4 is transfer length (if required)/parameter list length (if required)/allocation length (if required); Byte 5 is control code.

6. Data. The data phase covers both the data-in and data-out phases. In the data-in phase, the target requests that data be sent to the initiator. For this purpose, it asserts the I/O signal and negates the C/D and MSG signals during the REQ/ACK handshake(s) of the phase. In the data-out phase, it requests that data be sent from the initiator to the target. The target negates the C/D, I/O, and MSG signals during the REQ/ACK handshake(s) of this phase.
7. Message. The message phase covers both the message-out and message-in phases. The first byte transferred in either of these phases can be either a single-byte message or the first byte of a multiple-byte message. Multiple-byte messages are contained completely within a single message phase. The message system allows the initiator and the target to communicate over the interface connection. Each message can be one, two, or more bytes in length. In a single message phase, one or more messages can be transmitted (but a message cannot be split between multiple message phases).
8. Status. The status phase allows the target to request that status information be sent from the target to the initiator. The target asserts the C/D and I/O signals and negates the MSG signal during the REQ/ACK handshake(s) of this phase. The status phase normally occurs at the end of a command (although in some cases it may occur before transferring the command descriptor block).

14.2.3 I/O connectors

Serial data communication implies that the bits of a character (or byte) are transmitted consecutively to a receiver that assembles the bits back into a character. Data rate, error checking, handshaking, and character framing (start and stop bits) are pre-defined and must correspond at both the transmitting and receiving terminals.

Both serial synchronous and asynchronous transmissions can be implemented with a series of Recommended Standards (RS), which usually define signal levels, maximum bandwidth, connector pinout, supported handshaking signals, drive capabilities, and electrical characteristics of the serial transmissions.

The following briefly describes some common serial transmission standards. All these interfaces accept a range of electrical and physical parameters and may even operate in excess of the specified standard. The full specification for each standard is available from almost any supplier of engineering documents.

(1) RS-232

In the RS-232 standard, data are sent as a time-series of bits. Both synchronous and asynchronous transmissions are supported by this standard. In addition to the data circuits, the standard defines a number of control circuits used to manage the connection between the DTE and DCE. Each circuit only operates in one direction; that is, signaling from a DTE to the connected DCE or the reverse. Since transmit data and receive data use separate circuits, the RS-232 interface can operate in a full duplex manner, supporting concurrent data flow in both directions. The standard does not define character framing within the data stream, or character encoding.

1. Voltage levels. The RS-232 standard defines the voltage levels that correspond to logical "1" and logical "0" levels for the data and the control signals. Valid data and control signals are plus or

minus 3 to 15 V; however, the range near zero volts is not a valid RS-232 level. The standard specifies a maximum open-circuit voltage of 25 V: signal levels of ±5 V, ±10 V, ±12 V, and ±15 V are all commonly seen depending on the power supplies available within a device. RS-232 drivers and receivers must be able to withstand indefinite short-circuit to ground, or to any voltage level up to ±25 V. The slew rate, or how fast the signal changes between levels, is also controlled.

2. Connectors. The RS-232 standard specifies 20 different signal connections. Since most devices use only a few signals, smaller connectors can often be used. The standard recommended, but did not make mandatory, the D-subminiature 25-pin (DB-25) connector. In general, and according to the standard, terminals and computers have male connectors with DTE pin functions, and modems have female connectors with DCE pin functions. Other devices may have any combination of connector gender and pin definitions. Many terminals were manufactured with female terminals but were sold with a cable with male connectors at each end; the terminal with its cable satisfied the recommendations in the standard.
3. Cables. The RS-232 standard does not define a maximum cable length, but instead defines the maximum capacitance that a compliant drive circuit must tolerate. A widely used rule-of-thumb indicates that cables more than 50 feet (15 meters) long will have too much capacitance, unless special cables are used. By using low-capacitance cables, full speed communication can be maintained over distances of up to about 1000 feet. Other signal standards are better suited to longer distances.
4. Pinouts and signals. Commonly used RS-232 pinouts and signals are given in Table 14.3. Note that only the DB-25 connector pinouts are given in Table 14.3. Other types of connector, DE-9 (TIA-574), EIA/TIA 561, and Yost, are not given in this table.
5. Conventions. For functional communication through a serial port interface, conventions of bit rate, character framing, communications protocol, character encoding, data compression, and error detection, not defined in the RS-232 standard, must be agreed to by both sending and receiving equipment. For example, consider the serial ports of the original IBM PC. This implementation used an 8250 UART (universal asynchronous receiver-transmitter) using asynchronous start-stop character formatting with 7 or 8 data bits per frame, usually ASCII character encoding, and data rates programmable between 75 bps and 115,200 bps. Data rates above 20,000 bps are out of the scope of the standard, although they are sometimes used in commercially manufactured equipment.

(2) RS-422

RS-422, unlike RS-232, is a differential interface that defines voltage levels, and driver and receiver electrical specifications. On this interface, logic levels are defined by the difference in voltage between a pair of outputs or inputs. In contrast, a single-ended interface, for example RS-232, defines the logic levels as the difference in voltage between a single signal and a common ground connection. Differential interfaces are typically more immune to noise or voltage spikes.

Differential interfaces also have greater drive capabilities, which allow for longer cable lengths. RS-422 is rated up to 10 Mbps and can have cabling 4000 feet long. It also defines driver and receiver electrical characteristics that will allow one driver and up to 32 receivers on the line at once. RS-422 signal levels range from 0 to +5 V. RS-422 does not define a physical connector.

Table 14.3 Typical RS-232 Connector Pinouts and Signals

Signals	Name	Pin # (DB-25)	Function
GND	Ground	7	Common ground signal
PG	Protective ground	1	Protective ground signal
RxD	Received data	3	Data sent from DCE to DTE
TxD	Transmitted data	2	Data sent from DTE to DCE
DTR	Data terminal ready	20	Asserted by DTE to indicate that it is ready to be connected
DSR	Data set ready	6	Asserted by DCE to indicate it is ready to receive commands or data from the DTE
RTR	Ready to receive	4 (if hand-shaking)	Asserted by DTE to indicate to DCE that DTE is ready to receive data
RTS	Request to send	4	Asserted by DTE to prepare DCE to receive data
CTS	Clear to send	5	Asserted by DCE to acknowledge RTS for DTE to transmit
DCD	Carrier detect	8	Asserted by DCE when a connection is established with remote equipment

(3) RS-485

Similar to the RS-422 standard, RS-485 is also a differential interface, allowing cable lengths up to 4000 feet and data rates up to 10 Mbps. The signal levels are the same as those defined by RS-422, having electrical characteristics that allow for 32 drivers and 32 receivers to be connected to one line. This interface is ideal for multiple-drop or network environments. RS-485's triple-state driver (not dual-state) will allow the electrical presence of the driver to be removed from the line. The driver is in a triple-state or high impedance condition when this occurs. Only one driver may be active at a time and the other driver(s) must be triple-stated.

The output modem-control signal "Request to Send (RTS)" controls the state of the driver. Some communication software packages refer to RS-485 as RTS enable or RTS block-mode transfer.

RS-485 can be cabled in two ways: two-wire and four-wire mode. Two-wire mode does not allow for full-duplex communication. Two-wire mode requires that data be transferred in only one direction at a time and the two transmit pins should be connected to the two receive pins (Tx+ to Rx+ and Tx− to Rx−). Four-wire mode will allow full-duplex data transfers. RS-485 does not define a physical connector, a connector pinout, or a set of modem control signals.

(4) RS-499 and RS-530

Both RS-499 and RS-530 are similar to RS-422 and RS-485, in that they are differential interfaces, but these latest two standards provide a specified pinout defining a full set of modem-control signals that can be used for regulating flow control and line status. RS-449 is defined on a standard DB-37 (D-subminiature 37-pin) connector; RS-530 is backwards-compatible and replaces RS-449. RS-530 is

defined on a DB-25 connector. These two interfaces do not define an electrical specification, but they do provide a means of selecting a standard cabling interface.

14.3 DATA TRANSMISSION CONTROL DEVICES

Within a microprocessor chipset, a computer motherboard, or a multiple-core processor chipset, data signals are transferred internally using a parallel format: all the bits of a byte or a memory word are exchanged simultaneously among registers, buses, ASIC etc. However, for the data to be communicated over a serial channel, it must be converted from parallel to a serial bit stream. Some special hardware units including circuits or other devices are therefore required to control the mutual translations between serial and parallel formats.

This section lists a group of important hardware units for digital data transmission control. One of these is the universal receiver-transmitter, which comes in three types: universal asynchronous receiver-transmitter (UART), universal synchronous receiver-transmitter (USRT), and universal synchronous/asynchronous receiver-transmitter (USART). Other hardware units for data transmission control are multiplexers and modems. All of them may be built into a computer or programmable controller, added as components of an I/O interface board, or may consist of a single ASIC chip. Figure 14.14 shows an example application.

14.3.1 Universal asynchronous receiver-transmitter (UART)

A UART is a microchip or integrated circuit with a hardcoded program that controls the interface between a computer or programmable controller and its attached serial devices. Specifically, it provides the computer with an RS-232-specified DTE interface so that it can "talk" to and exchange data with modems and other serial devices. UART is commonly used with RS-232 for embedded systems communications. Many microprocessor chips thus provide functionality to convert UART to RS-232 signals.

As part of this interface, the main functions of a UART include the following: (1) convert the bytes received from the computer along parallel circuits into a single-serial bit-stream for outbound

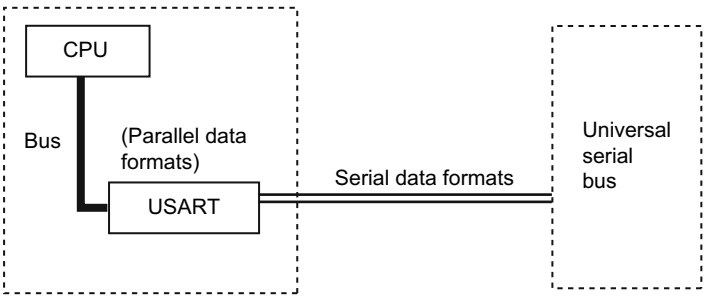


FIGURE 14.14

An illustration of the application of data transmission control circuits.

transmission; (2) on inbound transmission, convert the serial bit-stream into the bytes that the computer handles; (3) add a parity bit (if it has been selected) on outbound transmissions and check the parity of incoming bytes (if selected) and discard the parity bit; (4) add start and stop delineators on outbound data and strip them from inbound transmissions; (5) handle interrupts from serial devices with special ports such as keyboard and mouse; (6) possibly handle other kinds of interrupt and device management that require coordinating the computer's speed of operation with device speeds.

Figure 14.15 is a schematic diagram of a UART microchip, which usually contains the following components: (1) transmit and receive buffer, (2) transmit and receive register, (3) data bus buffer, (4) control logics, and (5) clock circuit. The control logic circuitry of UART has a number of internal registers that contain all the relevant status information for the device. These registers are generally mapped onto the computer's memory and can be accessed from the main CPU. The key is that each UART contains a shift register that is the fundamental method of conversion between serial and parallel forms.

The word asynchronous indicates that a UART recovers character timing information from the data stream, using designated start and stop bits to indicate the framing of each character. By convention, teletype-style UARTs send a start bit, 5 8 data bits, least-significant-bit first, an optional parity bit, and then a stop bit. The start bit is the opposite polarity of the data-line's normal state. The stop-bit is the data-line's normal state, and provides a space before the next character can start. In mechanical teletypes, the stop bit was often stretched to two bit times to give the mechanism more time to finish printing a character. A stretched stop bit also helps resynchronization. The parity bit can either make the number of bits odd or even, or it can be omitted. Odd parity is more reliable because it ensures that there will always be a data transition, and this permits many UARTs to resynchronize.

As shown in Figure 14.15, most UARTs are designed to assert an interrupt line when data have been received. This enables the CPU to run an interrupt service routine that removes data from the volatile

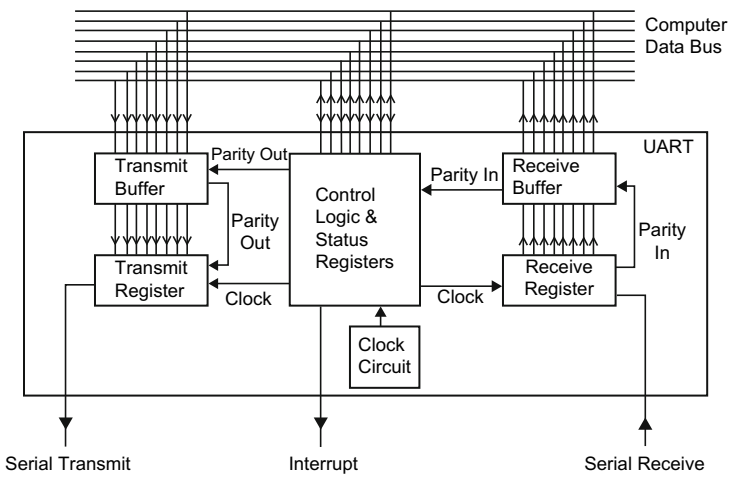


FIGURE 14.15

Schematic diagram of the UART microchip.

registers and buffers in the UART and places them into a more stable area of general-purpose computer memory.

14.3.2 Universal synchronous receiver-transmitter (USRT)

For synchronous transmission schemes, the USRT is used for conversion between serial and parallel data formats. A schematic is shown in Figure 14.16. In principle, the USRT is similar to the UART, except that the incoming data are clocked into the receive register with a signal that is derived (extracted) from the data themselves.

In synchronous transmission, the clock data are recovered separately from the data stream and no start/stop bits are used. This improves the efficiency of transmission on suitable channels; more of the bits sent are data. Therefore, the USRT is a circuit capable of receiving and sending data without requiring a start or a stop bit code, unlike the asynchronous procedure mentioned in the last subsection.

14.3.3 Universal synchronous/asynchronous receiver-transmitter (USART)

The universal synchronous asynchronous receiver-transmitter (USART) microchip is composed of some logic circuit, connected by an internal data bus with a microprocessor or a CPU. Figure 14.17 is a diagram of the pinouts of a USART microchip.

The USART is programmable, meaning the microprocessor or CPU can control its mode of operation using data bus control and command words. The USART microchip is typically architected into the following five control circuits:

1. Read/write control. The read/write control logic circuit accepts control signals from the control bus and command or control words from the data bus. The USART is set to an idle state by the RESET

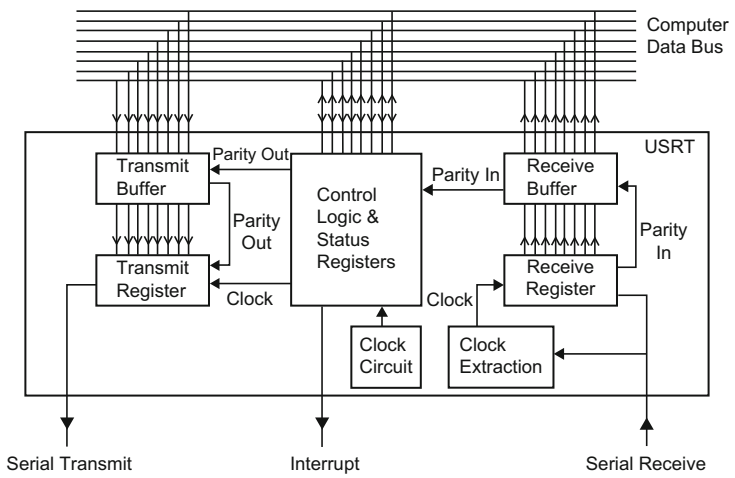
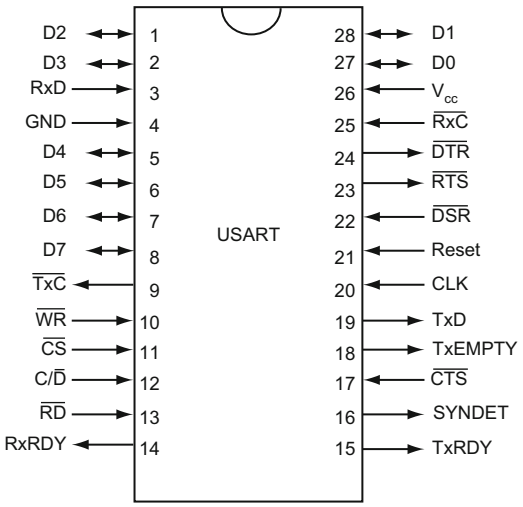


FIGURE 14.16

Schematic diagram of the USRT microchip.



D0 D7	Data bus (8 bits)
C/D	Control/data
RD	Read
WR	Write
CS	Chip select
CLK	Clock pulse (TTL)
Reset	Reset
TxC	Transmitter clock
TxD	Transmitter data
RxC	Receiver clock
RxD	Receiver data
RxRDY	Receiver ready
TxRDY	Transmitter ready
DSR	Data set ready
DTR	Data terminal ready
SYNDET	Sync detect
RTS	Request to send data
CTS	Clear to send data
TxEMPTY	Transmitter empty
Vcc	+s volt supply
GND	Ground

FIGURE 14.17

The pinouts of an USART microchip.

signal or control word. When the USART is in idle state, a new set of control words is required to program it for the applicable interface. The read/write control logic circuit receives a clock signal (CLK) that is used to generate internal device timing.

2. Modem control. The modem control logic circuit generates or receives four control or status signals (DSR, DTR, RTS, and CTS) used to simplify modem interfaces.
3. Baud rate generator (BRG). The BRG supports both asynchronous and synchronous modes of the USART. It is a dedicated 8-bit or more baud rate generator. The SPBRG register controls the period of a free-running 8-bit timer. In asynchronous mode, bit BRGH in the TXSTA register also controls the baud rate. In synchronous mode, bit BRGH is ignored.
4. Transmit buffer/transmit control. The transmit control logic converts the data bytes stored in the transmit buffer into an asynchronous bit stream. The transmit control logic inserts the applicable start/stop and parity bits into the stream to provide the programmed protocol. A start bit is used to alert the output device, a printer for instance, to get ready for the actual character (bit). The signal is sent just before the beginning of the actual character coming down the line. A stop bit is sent to indicate the end of transmission. The parity bit is used as a means to detect errors; odd or even parity may be used. Figure 14.18 is the transmit block diagram for a USART microchip.
5. Receive buffer/receive control. The receive control logic accepts the input bit stream and strips the protocol signals from the data bits. The data bits are converted into parallel bytes and stored in the receive buffer until transmitted to the microprocessor. Figure 14.19 is the receive block diagram for a USART microchip.

USART can be configured into asynchronous (full duplex), synchronous-master (half duplex), and synchronous-slave (half duplex) modes. They are discussed below in reference to Figures 14.18 and 14.19.

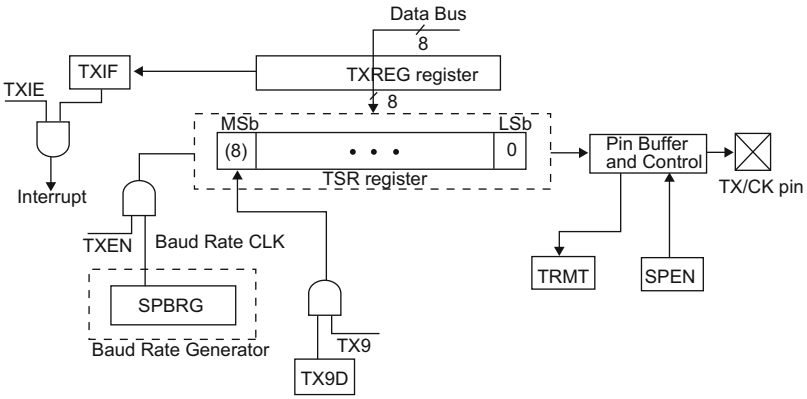


FIGURE 14.18

The USART transmit block diagram.

(1) USART asynchronous mode

In this mode, USART uses standard nonreturn-to-zero (NRZ) format (one start bit, eight or nine data bits, and one stop bit). The most common data format is 8 bits (of course there can be more bits than 8). An on-chip dedicated 8-bit baud rate generator can be used to derive standard baud rate frequencies from the oscillator. USART transmits and receives the LSb first. The transmitter and receiver are functionally independent but use the same data format and baud rate. The baud rate generator produces a clock either $\times 16$ or $\times 64$ of the bit shift rate, depending on the BRGH bit in the TXSTA register. Parity is not supported by the hardware, but can be implemented in software (stored as the ninth data

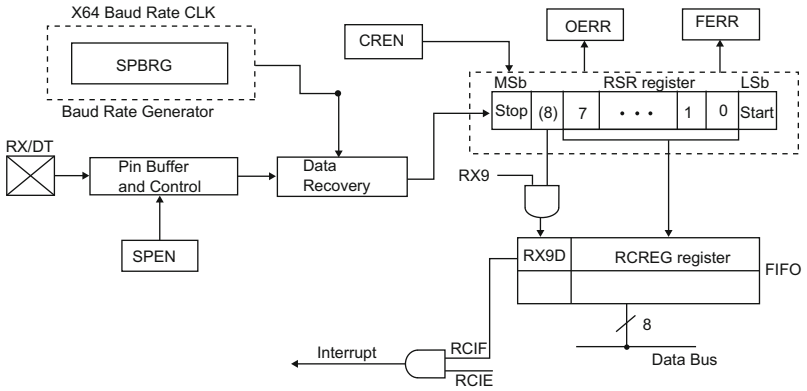


FIGURE 14.19

The USART receive block diagram.

bit). Asynchronous mode is stopped during SLEEP, and selected by clearing the SYNC bit in the TXSTA register.

The USART asynchronous module consists of the following hardware elements: baud rate generator, sampling circuit, asynchronous transmitter, and asynchronous receiver.

(a) USART asynchronous transmitter

The USART transmitter block diagram is shown in Figure 14.18. The heart of the transmitter is the (serial) transmit shift register (TSR). This obtains its data from the read/write transmit buffer, TXREG. The TXREG register is loaded with data in software. The TSR register is not loaded until the STOP bit has been transmitted from the previous load. As soon as this occurs, the TSR is loaded with new data from the TXREG register (if available). Once the TXREG register transfers the data to the TSR register (occurs in one timing cycle), the TXREG register is empty and the TXIF flag bit is set. This interrupt can be enabled/disabled by setting/clearing the TXIE enable bit. The TXIF flag bit will be set regardless of the state of the TXIE enable bit and cannot be cleared in software, but will reset only when new data are loaded into the TXREG register. While the TXIF flag bit indicates the status of the TXREG register, the TRMT bit in the TXSTA register shows the status of the TSR register. The TRMT status bit is a read-only bit that is set when the TSR register is empty. No interrupt logic is tied to this bit, so the user has to poll this bit to determine whether the TSR register is empty.

Transmission is enabled by setting the TXEN enable bit in the TXSTA register. Actual transmission will not occur until the TXREG register has been loaded with data and the baud rate generator (BRG) has produced a shift clock (Figure 14.18). Transmission can also be started by first loading the TXREG register and then setting the TXEN enable bit. Normally, when transmission is first started, the TSR register is empty, so a transfer to the TXREG register will result in an immediate transfer to TSR, resulting in an empty TXREG. A back-to-back transfer is thus possible.

(b) USART asynchronous receiver

The receiver block diagram is shown in Figure 14.19. The data are received on the RX/DT pin and drive the data recovery block. This is actually a high-speed shifter operating at $\times 16$ times the baud rate; whereas the main receive serial shifter operates at the bit rate or at 16 MHz.

Once asynchronous mode is selected, reception is enabled by setting the CREN bit in the RCSTA register. The heart of the receiver is the (serial) receive shift register (RSR). After sampling the RX/TX pin for the STOP bit, the received data in the RSR are transferred to the RCREG register (if it is empty). If the transfer is complete, the RCIF flag bit is set. The actual interrupt can be enabled/disabled by setting/clearing the RCIE enable bit. The RCIF flag bit is a read-only bit that is cleared by the hardware, when the RCREG register has been read and is empty. The RCREG is a double-buffered register, that is, it is a two-deep FIFO (first-in-first-out). It is possible for two bytes of data to be received and transferred to the RCREG FIFO and a third byte to begin shifting to the RSR register. On the detection of the STOP bit of the third byte, if the RCREG register is still full then the overrun error bit, OERR (RCSTA), will be set, and the word in the RSR will be lost. The RCREG register can be read twice to retrieve the two bytes in the FIFO. The OERR bit has to be cleared in software by resetting the receive logic (the CREN bit is cleared and then set). If the OERR bit is set, transfers from the RSR register to the RCREG register are inhibited, so it is essential to clear the OERR bit if it is set.

The framing error bit, FERR (RCSTA), is set if a stop bit is detected as a low level. The FERR bit and the ninth receive bit are buffered in the same way as the received data. Reading the RCREG will load the RX9D and FERR bits with new values, therefore it is essential for the user to read the RCSTA register before reading the next RCREG register so as not to lose the old (previous) information in the FERR and RX9D bits.

(2) USART synchronous master mode

In synchronous master mode, the data are transmitted in a half-duplex manner, in which transmission and reception do not occur at the same time. When transmitting data, reception is inhibited and vice versa.

Synchronous mode is entered by setting the SYNC bit in the TXSTA register. In addition, the SPEN enable bit in the RCSTA register is set to configure the TX/CK and RX/DT I/O pins to CK (clock) and DT (data) lines, respectively. The master mode indicates that the processor transmits the master clock on the CK line. The master mode is entered by setting the CSRC bit in the TXSTA register.

(a) USART synchronous master transmission

The USART transmitter block diagram is shown in Figure 14.18. The heart of the transmitter is the (serial) transmit shift register (TSR), which obtains its data from the read/write transmit buffer register TXREG. The TXREG register is loaded with data in software. The TSR register is not loaded until the last bit has been transmitted from the previous load, at which point, the TSR is loaded with new data from the TXREG (if available). Once the TXREG register transfers the data to the TSR register, it becomes empty and the TXIF interrupt flag bit is set. The interrupt can be set into enabled/disabled by setting/clearing enable the TXIE bit. The TXIF flag bit will be set regardless of the state of the TXIE enable bit and cannot be cleared in software. It will reset only when new data are loaded into the TXREG register.

While the TXIF flag bit indicates the status of the TXREG register, the TRMT bit in the TXSTA register shows the status of the TSR register. The TRMT bit is a read-only bit that is set when the TSR is empty. No interrupt logic is tied to this bit, so the user has to poll it to determine whether the TSR register is empty. The TSR is not mapped in data memory so it is not available to the user.

(b) USART synchronous master reception

Once synchronous mode is selected, reception is enabled by setting either the SREN bit or the CREN bit in the RCSTA register. Data are sampled on the RX/DT pin on the falling edge of the clock. If the SREN bit is set, then only a single word is received. If the CREN bit is set, the reception is continuous until it is cleared. If both bits are set, then the CREN bit takes precedence. After clocking the last serial data bit, the received data in the receive shift register (RSR) are transferred to the RCREG register (if it is empty). When the transfer is complete, the RCIF interrupt flag bit is set. The actual interrupt can be enabled/disabled by setting/clearing the RCIE enable bit. The RCIF flag bit is a read-only bit that is cleared by the hardware, in this case, when the RCREG register has been read and is empty. The RCREG is a double-buffered register, which means that it is a two-deep FIFO (first-in-first-out). It is possible for two bytes of data to be received and transferred to the RCREG register by FIFO and a third byte to begin shifting into the RSR register.

On the clocking of the last bit of the third byte, if the RCREG register is still full then the overrun error bit will be set.

(3) USART synchronous slave mode

Synchronous slave mode differs from the synchronous master mode in that the shift clock is supplied externally at the TX/CK pin (instead of being supplied internally in master mode). This allows the device to transfer or receive data while in SLEEP mode. Slave mode is entered by clearing the CSRC bit in the TXSTA register.

(a) USART synchronous slave transmit

The transmit operations of both the synchronous master and slave modes are identical except in the case of the SLEEP mode.

If two words are written to the TXREG register and then the SLEEP instruction is executed, the following will occur: (1) The first word will immediately transfer to the TSR register and transmit. (2) The second word will remain in the TXREG register. (3) The TXIF flag bit will not be set. (4) When the first word has been shifted out of TSR, the TXREG register will transfer the second word to the TSR and the TXIF flag bit will now be set. (5) If the TXIE enable bit is set, the interrupt will wake the chip from SLEEP and if the global interrupt is enabled, the program will branch to the interrupt vector.

(b) USART synchronous slave reception

The receive operations of both the synchronous master and slave modes are identical except in the case of the SLEEP mode. Also, bit SREN is a do-not-care in slave mode.

If receive is enabled, by setting the CREN bit, before the SLEEP instruction, then a word may be received during SLEEP. On completely receiving the word, the RSR register will transfer the data to the RCREG register and if the RCIE enable bit is set, the interrupt generated will wake the chip from SLEEP. If the global interrupt is enabled, the program will branch to the interrupt vector.

14.3.4 Bit-oriented protocol circuits

Bit-oriented protocols represent a class of data-link layer communication protocols that can transmit frames regardless of frame content. Unlike byte-oriented protocols, bit-oriented protocols provide full-duplex operation and are more efficient and reliable. However, the byte-oriented protocols use a specific character from the user character set to delimit frames in data-link communications. Today, byte-oriented protocol circuits have largely been replaced by bit-oriented protocols.

In current applications, there are two types of controller that can perform the bit-oriented protocol: the SDLC (synchronous data-link control) controller and the HDLC (high-level data-link control) controller. Figure 14.20 is a block diagram for both of these.

(1) SDLC controller

The SDLC controller is a bit-oriented protocol that presupposes an IBM protocol for use in a system network architecture (SNA) environment.

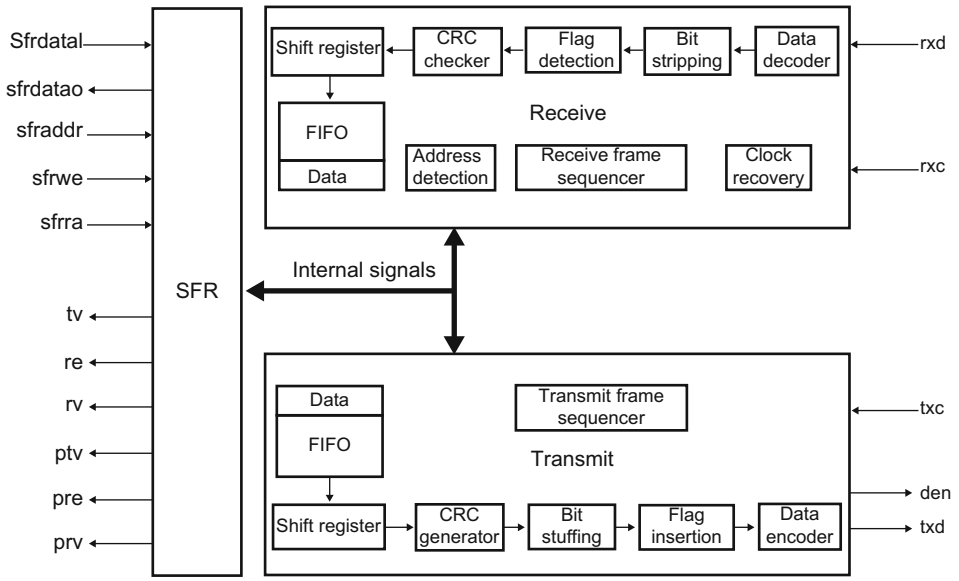


FIGURE 14.20

A function block diagram for SDLC and HDLC controllers.

It supports a variety of link types and topologies, and can be used with point-to-point and multi-point links, bounded and unbounded media, half-duplex and full-duplex transmission facilities, and circuit-switched and packet-switched networks.

The SDLC frame, shown in Figure 14.21, is bounded by a unique flag pattern. The address field always contains the address of the secondary involved in the current communication. Because the primary is either the communication source or destination, there is no need to include its address: it is already known by all secondaries. The control field uses three different formats, depending on the type of SDLC frame used. They are described as follows:

(a) Information (I) frames. These frames carry upper-layer and some control information. Send and receive sequence numbers and the poll final (P/F) bit perform flow and error control. The send sequence number refers to the number of the frame to be sent next. The receive sequence number provides the number of the frame to be received next. Both sender and receiver maintain send and receive sequence numbers. The primary uses the P/F bit to tell the secondary whether it requires an immediate response, and the secondary then uses this bit to tell the primary whether the current frame is the last in its current response.

(b) Supervisory (S) frames. These frames provide control information. They request and suspend transmission, report on status, and acknowledge the receipt of I-frames. They do not have an information field.

(c) Unnumbered (U) frames. These frames, as the name suggests, are not sequenced, and are used for control purposes, for example, to initialize secondaries. Depending on the function of the unnumbered frame, its control field is 1 or 2 bytes. Some unnumbered frames have an information field.

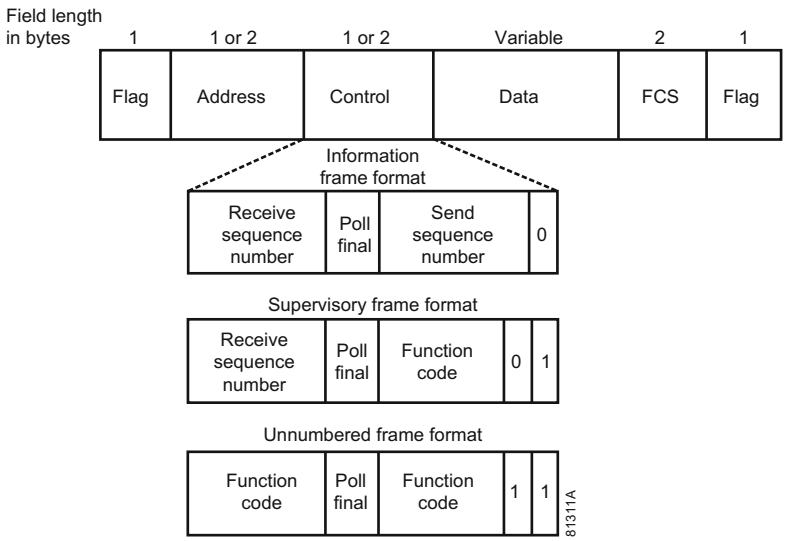


FIGURE 14.21

The SDLC frame format.

The frame check sequence (FCS) precedes the ending flag delimiter. This is usually a cyclic redundancy check (CRC) calculation remainder. The CRC calculation is redone in the receiver. If the result differs from the value in the sender's frame, an error is assumed.

(2) HDLC controller

The HDLC controller is an ISO communications protocol used in X.25 packet switching networks. It is a bit-oriented data-link control procedure under which all data transfer takes place in frames. Each frame ends with a frame check sequence for error detection.

HDLC shares the frame format of SDLC, and HDLC fields provide the same functionality as those in SDLC. HDLC also supports synchronous, full-duplex operation. HDLC differs from SDLC in several minor ways. First, HDLC has an option for a 32-bit or more checksum, and it does not support the loop or hub go-ahead configurations.

The major difference between the two is that SDLC supports only one transfer mode, while HDLC supports three. The three HDLC transfer modes are as follows:

(a) Normal response mode (NRM). This transfer mode is used by SDLC. In this mode, secondaries cannot communicate with a primary until the primary has given permission.

(b) Asynchronous response mode (ARM). This transfer mode allows secondaries to initiate communication with a primary without receiving permission.

(c) Asynchronous balanced mode (ABM). ABM introduces the combined node. A combined node can act as a primary or a secondary, depending on the situation. All ABM communication is between multiple combined nodes. In ABM environments, any combined station may initiate data transmission without permission from any other station.

14.3.5 Multiplexers

A multiplexer, sometimes simply referred to as "mux", is a device that selects between a number of input signals. In its simplest form, it will have two signal inputs, one input control, and one output. An everyday example of a multiplexer is the source selection control on a home stereo unit.

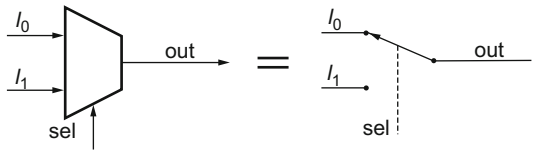
Multiplexers are used in building digital semiconductors such as CPUs and graphics controllers. In these applications, the number of inputs is generally a multiple of 2 (2, 4, 8, 16, etc.), the number of outputs is either 1 or a relatively smaller multiple of 2, and the number of control signals is related to the combined number of inputs and outputs. For example, a 2-input, 1-output multiplexer requires only one control signal to select the input, while a 16-input, 4-output multiplexer requires four control signals to select the input and two to select the output.

They are mainly categorized on the basis of the working mechanisms described in subsection 14.1.2. Of the available types, the time-division multiplexer is more complex, and the digital multiplexer is more important in applications.

(1) Digital multiplexer

In the designations of digital integrated circuits, the multiplexer is a device that has multiple input streams and only one output stream. It forwards one of the input streams to the output stream based on the values of one or more selection inputs or control inputs. For example, a digital multiplexer with two inputs is a simple connection of logic gates whose output is either input I_0 or input I_1 depending on the value of a third input Sel which selects the input. Its Boolean equation is "Out = (I_0 and Sel) or (I_1 and not Sel)". The logic of this multiplexer can be expressed as Figure 14.22(a), and its truth table is given in Figure 14.22(b).

(a)

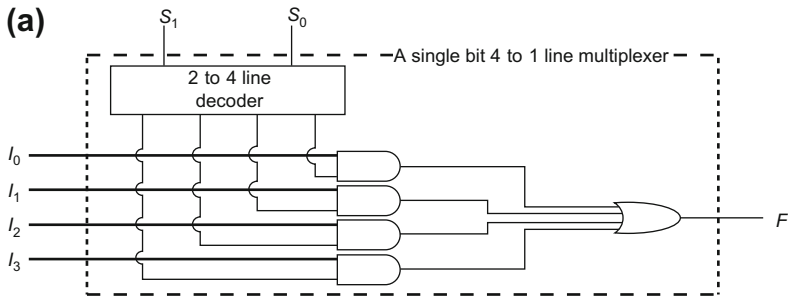


(b)

I_0	I_1	Sel	Out
0	0	0	0 (pick B)
0	0	1	0 (pick A)
0	1	0	1 (pick B)
0	1	1	0 (pick A)
1	0	0	0 (pick B)
1	0	1	1 (pick A)
1	1	0	1 (pick B)
1	1	1	1 (pick A)

FIGURE 14.22

A two-input digital multiplexer: (a) circuit principle, (b) truth table.



(b)

S_1 S_0	I_3	I_2	I_1	I_0	F
0 0	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	0
	0	0	1	1	1
	0	1	0	0	0
	0	1	0	1	1
	0	1	1	0	0
	0	1	1	1	1
	1	0	0	0	0
	1	0	0	1	1
	1	0	1	0	0
	1	0	1	1	1
	1	1	0	0	0
	1	1	0	1	1
	1	1	1	0	0
	1	1	1	1	1

S_1 S_0	I_3	I_2	I_1	I_0	F
0 1	0	0	0	0	0
	0	0	0	1	0
	0	0	1	0	1
	0	0	1	1	1
	0	1	0	0	0
	0	1	0	1	0
	0	1	1	0	1
	0	1	1	1	1
	1	0	0	0	0
	1	0	0	1	0
	1	0	1	0	1
	1	0	1	1	1
	1	1	0	0	0
	1	1	0	1	0
	1	1	1	0	1
	1	1	1	1	1

S_1 S_0	I_3	I_2	I_1	I_0	F
1 0	0	0	0	0	0
	0	0	0	1	0
	0	0	1	0	0
	0	0	1	1	0
	0	1	0	0	1
	0	1	0	1	1
	0	1	1	0	1
	0	1	1	1	1
	1	0	0	0	0
	1	0	0	1	0
	1	0	1	0	0
	1	0	1	1	0
	1	1	0	1	1
	1	1	0	1	1
	1	1	1	0	1
	1	1	1	1	1

S_1 S_0	I_3	I_2	I_1	I_0	F
1 1	0	0	0	0	0
	0	0	0	1	0
	0	0	1	0	0
	0	0	1	1	0
	0	1	0	0	0
	0	1	0	1	0
	0	1	1	0	0
	0	1	1	1	0
	1	0	0	0	1
	1	0	0	1	1
	1	0	1	0	1
	1	0	1	1	1
	1	1	0	0	1
	1	1	0	1	1
	1	1	1	0	1
	1	1	1	1	1

FIGURE 14.23

A single bit 4-to-1 line digital multiplexer: (a) circuit principle, (b) truth table.

Larger digital multiplexers are also common. Figure 14.23 is a single bit 4-to-1 line digital multiplexer. Its logic is also expressed by the circuit diagram in Figure 14.23(a), and the truth table in Figure 14.23(b).

Conversely, demultiplexers take one data input and a number of selection inputs, and they have several outputs. They forward the data input to one of the outputs, depending on the values of the selection inputs. Demultiplexers are sometimes convenient for designing general-purpose logic, because if the demultiplexer's input is always true, the demultiplexer acts as a decoder. This means that any function of the selection bits can be constructed by logically OR-ing the correct set of outputs.

(2) Time-division multiplexer

Time-division multiplexers (TDMs) share transmission time on an information channel among many data sources. Performance specifications for TDMs include number of channels, maximum data rate, wavelength range, operating voltage, optical output, electrical output, data transmission type, and data interface. Additional features may also be available.

One of the key factors in choosing a TDM is the selection of the transfer mode to be used. In synchronous transfer mode, data signals are sent at precise intervals that are regulated by a system clock. Additional start and stop pulses are not required. Asynchronous transfer mode (ATM) is a connection-oriented protocol that uses very short, fixed-length (53 bytes) packets called cells to carry voice, data, and video signals. By using a standard cell size, ATM can use software for data switching, and so can route and switch traffic at higher speeds. An asynchronous and/or synchronous time division multiplexer is capable of both asynchronous and synchronous transfer modes.

There are three cable choices for TDMs. Single-mode optical-fiber cable allows only one mode to propagate. The fiber has a very small core diameter of approximately 8 μm. It permits signal transmission at extremely high bandwidths and allows very long transmission distances. Multimode fiber-optic cable supports the propagation of multiple modes. It may have a typical core diameter of 50 μm with a refractive index that is graded or stepped, so allowing use of inexpensive LED light sources. Connector alignment and coupling is less critical than with single-mode fiber. Distances of transmission and transmission bandwidth are also less than with single-mode fiber due to dispersion. Single-mode/multimode time-division multiplexers can be used with both single-mode and multimode cable types.

Problems

1. Please think about why it is "over a physical distance" rather than "over a geographic distance" in this sentence: "The term data transmission concerns the transmission of digital electric or electromagnetic signals from the source electronic devices to destination electronic devices through some electric media over a physical distance".
2. Please investigate the correctness of this assertion: "the electrical signals transmitted by the cables or wires connecting the equipment in industrial control systems must be analog signals".
3. To send a digital data file between a source and a destination the asynchronous technique is used at the baud rate 128 bps. Suppose that you have a data file of 32 characters and each character is of 8 bits (one byte), please plot the pulse diagram of 3 seconds length for the data channel. (Note: suggest you choose the ASCII codeword standard; and remember that you can design the parity, start, and stop bits for each character; and checksum numbers for this file.)
4. Please redo problem 3 above with but using the EBCDIC codeword standard instead of the ASCII.
5. Please give your definition for lossless and lossy compression algorithms, respectively; then judge whether the Manchester coding algorithm is a lossless or lossy compression algorithm.
6. Can buffer circuits be used as one measure to avoid bus contention?
7. Please investigate the market to find all the bus interface and bus driver circuits used for data transmission between a CPU chip and peripherals over reasonable distances.
8. Can you make some modifications to the deployment of the balance circuit in Figure 14.4 to improve the capabilities of noise canceling for data transmissions in either a simple extension cable or a LAN?
9. Please explain why frequency division multiplexing (FDM) can be used only for analog data transmission and time division multiplexing (TDM) only for digital data transmission.
10. Please plot some diagrams to explain the working principle for dense wavelength division multiplexing (DWDM).
11. Please analyze the differences between I/O buses and I/O ports, between I/O buses and I/O connectors, and between I/O ports and I/O connectors; and then give your definition of I/O buses, I/O ports and I/O connectors.
12. Check all the ports and peripheral adaptors of one computer (desktop or laptop), such as USB ports, CD/DVD reader/writer, LAN modem, and Firewire, etc. Then plot the function block diagram of its motherboard system assuming it uses a PCI based platform.
13. In the above problem, please analyze the possible steps in the PCI bus configuration process in this computer on power on or restart.

14. In reference to Chapter 5 on microprocessor units, investigate the correctness of this assertion: within a microprocessor unit such as a CPU, the digital data signals are internally transferred using a parallel format: all the bits of a byte or a memory word are exchanged simultaneously among registers, buses, ASIC, and other components.
15. UART and USRT are certainly different in their receivers, which translate the received serial bits for character recovery into parallel bits: the UART works with asynchronous receiving, and the USRT with synchronous. Please investigate whether or not UART and USRT are different in their transmitters.
16. The USART can be configured into these six modes: asynchronous transmission, asynchronous receive, synchronous master transmission, synchronous master receive, synchronous slave transmission, and synchronous slave receive. Please analyze, based on Figure 14.18 and 14.19, why both asynchronous transmission and asynchronous reception may not require to be differentiated into master and slave modes?
17. Please investigate byte oriented protocol circuits to analyze why the byte oriented protocol circuits have largely been replaced by bit oriented protocols.
18. Please investigate the differences between SDLC and HDLC controllers in their protocols, circuits, and applications.
19. Based on the truth table in Figure 14.22, plot a real electronic (logic) circuit for this two input digital multiplexer.
20. Based on the truth table in Figure 14.23, plot a real electronic (logic) circuit for this single bit 4 to 1 line digital multiplexer.

Further Reading

David R. Smith. Digital Transmission Systems (3rd edition). Kluwer Academic. 2003.

John Park, Steve Mackay, Edwin Wright (Eds.). Practical Data Communications for Instrumentation and Control. Newnes. 2003.

Richard Zurawski (Ed.). The Industrial Communication Technology Handbook. CRC Press. 2005.

Christopher E. Strangio. Data communications basics. http://www.camiresearch.com/Data Com Basics/data com tutorial.html#anchor405943. Accessed: April 2009.

Dario J. Toncich. Data Communications & Networking for Manufacturing Industries. Chrystobel Engineering (Brighton, Australia). 1993.

Hong Ju Moon. Communication network for industrial automation. http://icat.snu.ac.kr:3333/rain e/intro/2.html. Accessed: April 2009.

GlobalSpec (www.globalspec.com). IC interface devices. http://semiconductors.globalspec.com/ProductFinder/Semiconductors Electronics/IC Interface Devices. Accessed: April 2009.

Kioskea (http://en.kioskea.net). Data transmission. http://en.kioskea.net/contents/transmission. Accessed: April 2009.

Altera (www.altera.com). Industrial connectivity. http://www.altera.com/end markets/industrial/ind index.html. Accessed: April 2009.

Wikipedia (http://en.wikipedia.org). Accelerated graphics port. http://en.wikipedia.org/wiki/Accelerated Graphics Port. Accessed: October 2009.

Wikipedia (http://en.wikipedia.org). SCSI. http://en.wikipedia.org/wiki/SCSI. Accessed: October 2009.

Wikipedia (http://en.wikipedia.org). RS 232. http://en.wikipedia.org/wiki/RS 32. Accessed: October 2009.

Wikipedia (http://en.wikipedia.org). IEEE 1394 interface. http://en.wikipedia.org/wiki/IEEE 1394 interface. Accessed: October 2009.

Wikipedia (http://en.wikipedia.org). Universal serial bus. http://en.wikipedia.org/wiki/Universal Serial Bus. Accessed: October 2009.

Wikipedia (http://en.wikipedia.org). Conventional PCI. http://en.wikipedia.org/wiki/Conventional PCI. Accessed: October 2009.

Actel (http://www.actel.com). PCI Arbiter Core. Version 4.0. January 2002.
Black Box (www.blackbox.co.uk). 2000. Single channel RS 232/530/422/485/20mA current loop interface. http://www.blackbox.co.uk/technical/manuals/I/IC601C%20Manual.pdf. Accessed: October 2009.
Black Box (www.blackbox.co.uk). 2000. Single channel async MIL 188. http://www.blackbox.co.uk/technical/manuals/I/IC603C%20Manual.pdf. Accessed: October 2009.
Trimex Marketing Inc. (www.trimex.com). RS 422. in Lava I/O News, 4(11). November 2002.
Pulsewan (www.pulsewan.com). Synchronous data link control and derivatives. http://www.pulsewan.com/data101/pdfs/sdlcetc.pdf. Accessed: October 2009.
Microchip (ww1.microchip.com) Section 18. USART. http://ww1.microchip.com/downloads/en/DeviceDoc/31018a.pdf. Accessed October 2009.

CHAPTER

Microprocessor boot code 15

At power up, a microprocessor unit needs initialization and self-test operations to load an operating system. Once the operating system is loaded, the microprocessor is ready to run application programs. Sometimes, an instruction is given to reboot or reset the operating system, i.e. to reload it. Booting or loading an operating system is different from installing it, which is generally an initial one-time activity, which stores the operating system source code on hard disk, where it is ready to be booted (loaded) into random access memory. The microprocessor unit's storage is closer to the microprocessor and faster to work with than the hard disk, so when an operating system is installed, it is usually set up so that when the microprocessor unit is turned on, the system is automatically booted as well. If storage (memory) runs out, or the operating system or an application program encounters an error, it may display an error message or the screen may freeze. In this event, the operating system may need to be rebooted.

A boot or reboot system in a single multiprocessor makes use of central processing units (CPUs) of different types. On larger units (including mainframes), the equivalent term for boot is initial program load and for reboot it is reinitial program load. The booting of an operating system works by loading a very small program into the microprocessor set and then giving that program control so that it in turn loads the entire operating system. These operations routinely involve storing the configuration table, initialization self-test code, and loading the operating system specific to each processor. The multiprocessor system automatically transfers initialization operations from a default processor to the first alternative if the default fails, and automatically transfers initialization operations to a second alternative processor if the first fails, and so forth, depending upon how many processors are installed.

15.1 CODE STRUCTURES

The boot code of a microprocessor unit is a complex program set consisting mainly of BIOS and kernel, master boot record (MBR), and boot program.

15.1.1 BIOS and kernel

(1) BIOS

In computing, BIOS stands for basic input and output system, and refers to the software code run by a computer or a programmable controller when first powered on. Its primary function is to prepare the machine so other software programs can load, execute, and assume control. The other main responsibilities of the BIOS include booting the system, and providing the BIOS setup program that allows the changing of BIOS parameters.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10015-4

When a computer or a controller is first turned on, the microprocessor needs some instructions for execution. However, since the machine has just been turned on, all the registers of its system memory are empty; there are no programs to run. To make sure that the BIOS program is always available to the microprocessor, even when first turned on, it is hard-wired into a read-only memory (ROM) chip on the system's motherboard. A uniform standard was created between the makers of microprocessors and the makers of BIOS programs to ensure that the microprocessor would always look in the same place in memory to find the start register of the BIOS program. The microprocessor gets its first instructions from this location, and the BIOS program then begins executing. The BIOS program then begins the system boot sequence that calls other programs, and gets the operating system loaded, and the computer or controller can start running.

A microprocessor system can contain several BIOS firmware chips, particularly multiprocessor systems for supercomputers and programmable industrial controllers. The motherboard BIOS typically contains code to access fundamental hardware components such as the keyboard, ATA (IDE) hard disk controllers, USB ports, graphical user interfaces, and storage devices. The motherboard ensures that the instruction at the reset vector is a jump to the memory location mapped to the BIOS entry point. In addition, plug-in adapter cards such as SCSI, RAID, network interface cards, and video boards often include their own BIOS, complementing or replacing the system BIOS code for the given component. In some cases, where devices may also be used by add-in adapters, and actually directly integrated on the motherboard, the add-in ROM may also be stored as separate code on the main BIOS flash chip. It may then be possible to upgrade this add-in BIOS (sometimes called an option ROM) separately from the main BIOS code.

(2) Kernel

The kernel is a program that constitutes the central core of an operating system. It has complete control over everything that occurs in the system, and is the first part of the operating system to load into memory during booting, and it remains there for the entire duration of the session, since its services are required continuously. It is therefore important for it to be as small as possible while still providing all the essential services needed.

Because of its critical nature, the kernel code is usually loaded into a protected area of memory, which prevents it from being overwritten by other, less frequently used parts of the operating system, or by application programs. The kernel works in kernel space, whereas everything a user normally does, such as writing text or running programs is done in user space. This separation is made to prevent user data and kernel data from interfering with each other and thereby diminishing performance or causing the system to become unstable and possibly crash.

The kernel provides basic services for all other parts of the operating system, typically including memory, process file and I/O (input/output) management. These services are requested by other parts of the operating system or by application programs through a specified set of program interfaces referred to as system calls. The contents of a kernel vary considerably according to the operating system, but they typically include the following:

(a) a scheduler, which determines how the various processes share the kernel's processing time (including in what order);
(b) a supervisor, which grants use of the controller or computer to each process when it is scheduled;

(c) an interrupt handler, which handles all requests from the various hardware devices (such as disk drives and the keyboard) that compete for the kernel's services;
(d) a memory manager, which allocates the system's address spaces (i.e., locations in memory) among all users of the kernel's services;
(e) many (but not all) kernels also provide a "device I/O supervisor" category of services. These services, if available, provide a uniform framework for organizing and accessing the many hardware device drivers that are typical of an embedded system.

The kernel should not be confused with the BIOS. The latter is an independent program stored in a chip on the motherboard (the main circuit board of a computer or a controller) that is used during the booting process for such tasks as initializing the hardware and loading the kernel into memory. Whereas the BIOS always remains in the computer and is specific to its particular hardware, the kernel can be easily replaced or upgraded by changing or upgrading the operating system or, in the case of Linux, by adding a newer kernel or modifying an existing one.

Kernels can be classified into four broad categories: monolithic kernels, microkernels, hybrid kernels, and exokernels. Each has its own advocates and detractors.

When a computer or a programmable controller crashes, it actually means the kernel has crashed. If only a single program has crashed but the rest of the system remains in operation, then the kernel itself has not crashed. A crash is the situation in which a program, either a user application or a part of the operating system, stops performing its expected function(s) and wrongly responds to other parts of the system. The program might appear to the user to freeze. If such a program is critical to the operations of the kernel, the entire computer or controller could stall or shut down.

15.1.2 Master boot record (MBR)

When turning on a computer or a programmable controller, the processor attempts to begin processing data. But, since the system memory is empty, the processor does not really have anything to execute, or even begin to know where to look for it. To ensure that the microprocessor chipset will always boot regardless of the BIOS code, both chip and BIOS manufacturers developed their code so that the microprocessor, once turned on, always reads the first sector (sector zero) of the hard disk, which is called the master boot record (MBR), also referred to as the master boot sector or even just the boot sector. It then simply loads the contents of the MBR into memory and jumps to that location to start executing whatever code is in the MBR.

Similarly, every hard disk must have a consistent starting point where key information is stored about it, such as the number of partitions and what type they are. There also must be some place where the BIOS can load the initial boot program which starts the process of loading the operating system. The MBR is always located at cylinder 0, head 0, and sector 1; the first sector on the disk. This is the consistent starting point that the disk will always use, and is always consulted for instructions and information on how to proceed with the boot process and load the operating system.

As illustrated in Figure 15.1, the master boot record contains the following structures.

(a) Master partition table. This small bit of code, which is referred to as a table, contains a complete description of the partitions on the hard disk. When the developers designed the size of this master partition table, they left just enough room for the description of four partitions, hence the four partition (four physical partitions) limit. This is the reason that a hard disk may only have four

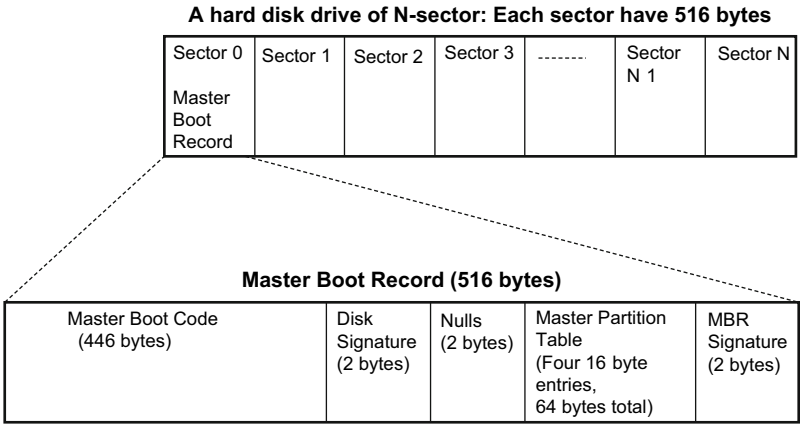


FIGURE 15.1

The layout of master boot record (MBR) on the hard disk.

true partitions, also called primary or physical partitions. Any additional partitions must be logical ones that are linked to (or are part of) one of the primary partitions. One of the partitions is marked as active, indicating that it is the one that the computer or the controller should used to continue the boot process.

(b) Master boot code. The master boot record is the small bit of computer code that the BIOS loads and executes to start the boot process. This code, when fully executed, transfers control to the boot program stored on the boot (active) partition, which in turn loads the operating system.

The MBR working in the boot process has bootstrapping firmware contained within the ROM. The MBR of a drive usually includes its partition table, which is used to load and run the boot record of the partition that is marked with the active flag. This design allows the BIOS to load any operating system without knowing exactly where to start inside its partition. Because the MBR is read almost as soon as the computer or controller starts, many computer viruses work here before the virus scanner software is activated, by changing the code within it. Technically, only partitioned media contain a master boot record, while unpartitioned media only have a boot sector as the first sector. In both cases, the BIOS transfers control to the first sector of the disk after reading it into memory. A legacy that the MBR can have is partition selection code, which loads and runs the boot (first) sector of the selected primary partition. That partition boot sector contains another boot loader. Newer MBRs, however, can directly load the next stage from an arbitrary location on the hard disk drive. This can pose some problems with dual-booting, as the boot loader whose location is coded into the MBR must be configured to load each operating system. If one operating system must be reinstalled, it may overwrite the MBR such that it will load a different boot loader.

15.1.3 Boot program

Figure 15.2 gives code examples that can be used to boot a CPU with a microprocessor, and run a program (operating system or an application code) starting at the bottom of the first bank of Flash.

This section of the code is copied from RAM into the top of EEPROM. In these code examples, the purpose is to allow easy booting of the CPU without having to preload the Flash banks with boot code. The Flash bank select addressing is indeterminate (messed up) following reset because port G defaults to input. The EEPROM is mapped to the top of the address space, and therefore contains the reset and interrupt vectors. The top 288 bytes of EEPROM are normally protected to minimize the possibility of accidentally making the module unbootable.

```
BOOT ORG 2000H                ; first 256 bytes of external RAM

; INITIAL CPU SETUP FOR LOADER

       LDS #01FFH             ; init stack
       LDX #1000H             ; register base address

       LDAA #10010001B        ; adpu, irqe, dly, cop = 65mS
       STAA OPTION,X

       LDAA #00000101B
       STAA CSCTL,X           ; enable program CS for 32K
       LDAA #00000000B
       STAA CSGADR,X          ; RAM starts at address 0000H
       LDAA #00000001B
       STAA CSGSIZ,X          ; RAM block size is 32K
       LDAA #00001111B
       STAA DDRG,X            ; bank select bits = outputs
       CLR PORTG,X            ; select 1ST bank


       JMP 8000H              ; point to APPLICATION in FLASH

; RESET & INTERRUPT VECTORS

; Make these point to the appropriate service routines within the
; application.

       ORG 20FFH-29H          ; Reset & Interrupt vectors for EEPROM

       DWM 0FF00H             ; SCI ; D6 SCI (FFD6)
       DWM 0FF00H             ; SPI ; D8 SPI
       DWM 0FF00H             ; PAIE ; DA PULSE ACCUMULATOR I/P EDGE
       DWM 0FF00H             ; PAO ; DC PULSE ACCUMULATOR OVERFLOW
       DWM 0FF00H             ; TOF ; DE TIMER OVERFLOW
       DWM 0FF00H             ; OC5 ; E0 TIMER O/P COMPARE 5
       DWM 0FF00H             ; OC4 ; E2 TIMER O/P COMPARE 4
       DWM 0FF00H             ; OC3 ; E4 TIMER O/P COMPARE 3
       DWM 0FF00H             ; OC2 ; E6 TIMER O/P COMPARE 2
       DWM 0FF00H             ; OC1 ; E8 TIMER O/P COMPARE 1
       DWM 0FF00H             ; IC3 ; EA TIMER I/P COMPARE 3
       DWM 0FF00H             ; IC2 ; EC TIMER I/P COMPARE 2
       DWM 0FF00H             ; IC1 ; EE TIMER I/P COMPARE 1
       DWM 0FF00H             ; RTI ; F0 REAL TIME INTERRUPT
       DWM 0FF00H             ; IRQ ; F2 EXTERNAL IRQ
       DWM 0FF00H             ; XIRQ   ; F4 EXTERNAL XIRQ
       DWM 0FF00H             ; SCI    ; F6 SOFTWARE INTERRUPT (SWI)
       DWM 0FF00H             ; ILLOP  ; F8 ILLEGAL OPCODE
       DWM 0FF00H             ; COP    ; FA COP OPERATED
       DWM 0FF00H             ; CLM    ; FC CLOCK MONITOR OPERATED
       DWM 0FF00H             ; START  ; FE RESET

BOOTEND                       ; end of boot code loaded into EEPROM
```

FIGURE 15.2

An example of a CPU boot program.

This code occupies the top 256 bytes of EEPROM, most of which is unused. The interrupt vectors are all forced to the boot program in EEPROM. These would normally point to interrupt service routines somewhere. The code examples in Figure 15.2 are written in the Motorola AS11 freeware assembler and are provided as examples only. There is no guarantee that the examples will work in a particular environment when applied.

15.2 SINGLE-PROCESSOR BOOT SEQUENCES

The following are the main steps in a typical boot sequence (or boot process). Of course, this will vary with hardware components, BIOS, and so on and especially with what peripherals are in the computer or controller system.

(1) Power on

The internal power supply for this microprocessor unit or CPU chipset turns on and initializes. There is some delay until it can generate reliable power for the rest of the computer, and having it turn on prematurely could potentially lead to damage. Therefore, the unit or chipset will generate a reset signal to the microprocessor until it receives the "power good" signal from the power supply.

(2) Load BIOS, MBR and boot program

When the microprocessor receives the reset signal, it will be ready to start executing an instruction, but there will be nothing at all in the memory to execute at first. Of course, the microprocessor makers know this will happen, so they preprogram the microprocessor to always look at the same place in the system BIOS in a specific ROM location for the start of the BIOS boot program. This is normally right at the end of the system memory, so that the size of the ROM can be changed without creating compatibility problems. Since, in most cases, there are only 16 bytes left from there to the end of conventional memory, this location just contains a jump instruction telling the microprocessor where to go to find the real BIOS start-up program.

(3) Initiate hardware components

The first thing that the BIOS does when it boots the system is to perform what is called the power-on self-test, or POST for short. The POST is a built-in diagnostic program that checks all hardware components in the system to ensure that everything is present and functioning properly, before the BIOS begins the actual boot. It later continues with additional tests (such as the memory test) as the boot process is proceeding.

The POST starts with an internal check of the CPU and of the boot code by comparing code at various locations against a fixed template. Then, the POST checks the bus, ports, system clock, display adapter memory, RAM, DMA, keyboard, floppy drives, hard drives, and so forth. The CPU sends signals over the system bus to make sure that these devices are functioning properly. In addition to the POST, the BIOS initialization routines initialize memory refresh, and load BIOS routines to memory. These routines will add to the system BIOS routines with routines and data from other BIOS chips on installed microcontrollers (note this is the microcontroller rather than an industrial controller). The routine then compares the information it has gathered with the

information stored in the setup program. If there are any discrepancies, it halts the boot process and informs the operator. If everything is satisfactory, this will usually be displayed to the operator on screen.

The POST runs very quickly, and the user will normally not even notice that it is happening. If any errors are detected, the system may deliver an error code. Error codes, both visual and audible, differ from manufacturer to manufacturer. To interpret them, the operator will need a table of these codes from the manufacturer of the system's motherboard. If any problems are encountered during the POST routine, then the operator knows that it is hardware-related. In some cases, the operator can find and repair problems by searching for poor connections, damaged cables, seized fans, and power problems; or the operator may need to check that adapter cards and RAM are installed properly in their respective slots. In extreme cases, the operator may have to strip the system down to its motherboard, RAM, video card, power supply, and CPU. By adding the rest of the devices individually and restarting after each one, the operator can sometimes discover the cause of the problem.

The BIOS often does more tests on the system, including the memory count-up test. It will generally display a text error message on the screen if it encounters an error at this point; these error messages and their explanations can be found in this part of the BIOS Troubleshooting Expert.

It also performs a system inventory of sorts, doing more tests to determine what sort of hardware is in the system, displaying a summary screen about the system's configuration. Checking this page of data in the screen can be helpful in diagnosing setup problems.

(4) Initiate interrupt vectors

The initialization of the system during POST creates interrupt vectors to the proper interrupt handling routines and sets up registers with parameters. In Figure 15.2, the interrupt vector table is included in the boot sector program, thus initializing the interrupt vectors to set up pointers (or registers) in memory to access those interrupt handling routines. In addition to the POST, interrupt vectors are reinitialized and system timers are reinitialized. In other words, the BIOS code initializes the computer or controller system to such a state that it is ready to load the operating system. Issuing an interrupt performs the loading of the operating system.

(5) Transfer to operating system

Whenever a computer or an industrial controller is turned on, BIOS takes control and performs a lot of operations. It checks the hardware, ports, and so on and finally it loads the MBR program into memory (RAM). At this point the MBR takes control of the booting process. When only one operating system is installed in the system, the functions of the MBR are as follows:

(a) the boot process starts by executing MBR code in the first sector of the hard disk;
(b) the MBR looks over the partition table to find the active partition;
(c) control is passed to that partition's boot record to continue booting;
(d) the partition's boot record locates the system-specific boot files;
(e) these boot files then continue the process of loading and initializing the rest of the operating system.

15.3 MULTIPROCESSOR BOOT SEQUENCES

In Chapter 5, the multi-core microprocessor unit, or multiprocessor chipset, was discussed in detail, including its architectures, function blocks, working mechanisms and technical specifications. This section will concentrate on the techniques related to booting and resetting such chipsets.

(1) Microprocessors in multiprocessor systems

To maintain compatibility with existing software products, the following is based on the Intel microprocessor family. Figure 15.3 gives a different point of view of a compliant system, showing the configuration of the CPUs of a multiprocessor system. While all CPUs in a compliant system are functionally identical, this figure classifies them into two types: the bootstrap processor (BSP) and the application processors (AP). Which processor is the BSP is determined by hardware or by hardware in conjunction with the BIOS. This differentiation is for convenience and is in effect only relevant during the initialization and shutdown processes. The BSP is responsible for initializing the system and for booting the operating system; APs are activated only after the operating system is up and running. CPU1 is designated as the BSP. CPU2, CPU3, and so on, are designated as the APs.

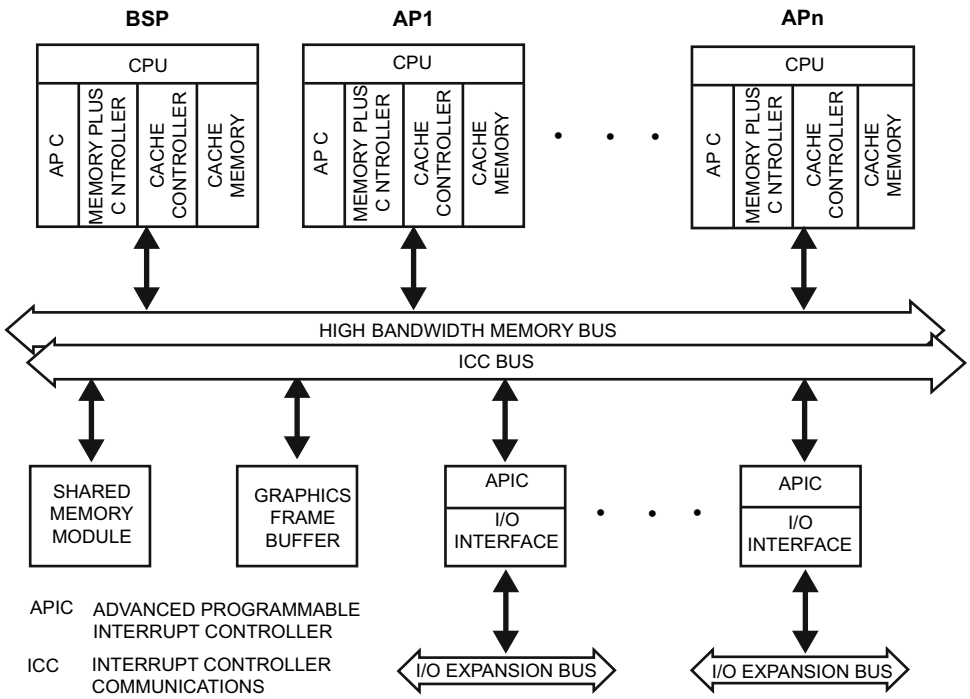


FIGURE 15.3

Architecture of the multiprocessor system based on an Intel microprocessor family.

A microprocessor-specific initialization is one of the basic support functions of a compliant multiprocessor system, along with processor start-up and shutdown. With it, the BSP can selectively initialize an AP for subsequent start-up, or recover from a fatal system error. This type of initialization function is exclusively used by the multiprocessor operating system or BIOS self-test routine. The system must be designed so that it can be performed by software programming; it should not be initiated by hardware.

(2) BIOS of a multiprocessor system

A BIOS always functions as an insulator between the hardware on one hand, and the operating system and applications software on the other hand. A standard uniprocessor BIOS (a) tests system components; (b) builds configuration tables to be used by the operating system; (c) initializes the microprocessor and the rest of the system to a known state; and (d) provides run-time device-oriented services.

For a multiprocessor system, the BIOS may perform the following additional functions: (a) pass configuration information to the operating system that identifies all microprocessors and other components of the system; (b) initialize all microprocessors and the rest of the components to a known state.

The multiprocessor chipset offers a wide range of capability in the BIOS. At the lower end of the capability scale, the system developer can simply insert a multiprocessing floating pointer structure in the standard BIOS. The cost of this level of simplicity in the BIOS, however, is that the system developer has less flexibility in the design of the hardware. At the higher end of the BIOS capability scale might be a BIOS that dynamically configures the system to provide resilience in the face of component malfunctions. Provision must be made to prevent all processors from executing the BIOS after a power-on reset. System developers may choose to do this by hardware alone, or by cooperation between hardware and the BIOS. In the latter case, the BIOS may be used for selecting the BSP and placing all APs to sleep after POST. It may use the advanced programmable interrupt controller (APIC) ID to identify each processor and select the proper code sequence to execute. Only the selected BSP continues to load the operating system after the POST routine.

(3) Operating system of a multiprocessor system

Two types of operating systems are able to utilize a multiprocessor: partitioned and symmetric multiprocessing. In partitioned architecture, each CPU boots into separate segments of physical memory and operates independently; in a symmetric system, microprocessors work in a shared space, executing threads within the operating system independently. This goal is achieved by allowing a flexible balance between the capabilities of the hardware and those of the BIOS. The potentially vast variety of hardware configurations is reduced by the BIOS to a few simple scenarios that can be readily handled by the low-level boot-up phase of the operating system.

One of the main functions of the system BIOS is to construct the multiprocessor floating pointer structure and the configuration table. Because this table is optional, the BIOS must set the multiprocessor feature information bytes in the multiprocessor floating pointer structure to indicate whether a multiprocessor configuration table is present. If the table is required, the BIOS constructs it in conjunction with the BSP and APs. The BIOS is responsible for synchronizing the activities of the APs during the construction of the table. It may need some synchronization during initialization so that each microprocessor may be brought up in the proper order. The mechanism for synchronization is not

specified; however, an example described in the following paragraphs uses AP status flags as a synchronization mechanism. This procedure also initializes the APs serially. System developers may employ other mechanisms, and may initialize all processors in parallel to minimize the system start-up time. The BIOS maintains an initialized AP status flag for each AP. Each AP will begin executing the same BIOS code as the BSP, but will eventually be put in a halt state or held in a loop until the BSP enables its AP status flag.

(4) Boot sequence of a multiprocessor system

Once system power is applied or the reset button is pressed (if the system is so equipped), a hardware circuit generates a system reset sequence to put all the system hardware into an initial state. All active microprocessors start to execute instructions and enter the POST (power-on-self-test) procedure of the BIOS, which initializes all components and constructs various system tables.

The goal of the hardware environment of a multiprocessor chipset is that a single, shrink-wrapped operating system can boot-up and fully utilize a wide variety of multiprocessor systems. The following explains how the operating system can handle (a) operating-system boot-up; (b) self-configuration; (c) interrupt mode initialization; (d) application processor start-up; (e) application processor shutdown; (f) dynamic interrupt masking; and (g) support for unequal processors.

While all microprocessors in a multiprocessing-compliant system are functionally identical, one of them will be designated as the bootstrap processor (BSP) at system initialization. The remainder are designated as the application processors (APs). The BSP is responsible for booting the operating system, but once this is running, the BSP functions as an AP. Usually a microprocessor is designated as the BSP because it is capable of controlling all system hardware, including AP start-up and shutdown. It is not necessarily the first microprocessor, especially in fault-tolerant multiprocessor systems in which this designation can be given to any available microprocessor.

When the first instruction of the operating system is executed, the APs can be the following states:

(a) the APs have been restrained (either by the BIOS or by the hardware) from executing operating system code;
(b) the APs are in a halted condition with interrupts disabled; this means that the AP's local APICs are passively monitoring the APIC bus and will react only to interprocessor interrupts.

The operating system's first task is to determine whether the system conforms to the multiprocessor specification made by the manufacturer. This is done by searching for the multiprocessor floating pointer structure. If detected, this indicates that the system is multiprocessor-compliant, and the operating system should continue to look for the multiprocessor configuration table. If the system is not multiprocessor-compliant, the operating system may attempt other means of multiprocessor system detection, if it is capable of doing so, or treat the system as a uniprocessor system.

The AP may be started either by the BSP or by another active AP. The operating system causes application processors to start executing their initial tasks by using a universal algorithm. This consists of a sequence of interprocessor interrupts and short programmatic delays to allow the APs to respond to the wakeup commands, and assumes that the BSP is starting an AP for documentation convenience.

Some older operating systems for multiprocessors cannot support microprocessors of different types, speeds, or capabilities. However, as microprocessor lifetimes increase and new generations of microprocessors arrive, the potential for dissimilarity among microprocessors increases. Most multiprocessor specifications addresses this possibility by providing a configuration table to help the

operating system configure itself. Operating system writers should factor in microprocessor variations, such as microprocessor type, family, model, and features, to arrive at a configuration that maximizes overall system performance. At a minimum, the multiprocessor operating system should remain operational and should support the common features of unequal processors.

Problems

1. Please list all the differences between booting and installing an operating system on a single processor chipset.
2. The boot code for a microprocessor chipset should be a program package consisting of BIOS, kernel, MBR, and boot program, which are stored in the system ROM. Please plot a possible layout of these programs in the system ROM.
3. Please plot function block diagrams for the kernel of a single processor and a multiprocessor chipset, respectively.
4. During the booting of a multiprocessor chipset, the AP may be started either by the BSP or by another active AP. The operating system causes application processors to start executing their initial tasks in the operating system code by using a universal algorithm. Please consult the Further Reading to explain this universal algorithm as far as possible.
5. Are there some differences between the kernel of a uniprocessor chipset and the kernel of a multiprocessor chipset?

Further Reading

PC Guide (http://www.pcguide.com). System boot sequence. http://www.pcguide.com/ref/mbsys/bios/boot.htm. Accessed: October 2008.

PC Guide (http://www.pcguide.com). PC guide. http://www.pcguide.com/index.htm. Accessed: October 2008.

Wikipedia (http://en.wikipedia.org). Multi core. http://en.wikipedia.org/wiki/Multi core. Accessed: October 2008.

Embedded Company (http://www.embedded.com). Multiprocessor. http://www.embedded.com/columns/technicalinsights/187202305? requestid=33986. Accessed: October 2008.

Gustavo Duarte (http://duartes.org/gustavo/blog/). How computers boot up. http://duartes.org/gustavo/blog/. October 2008.

Intel Corporation (http://www.intel.com). Multiprocessor Specification Version 2.2; 2007.

CHAPTER

Real-time operating systems 16

16.1 INTRODUCTION

An operating system is a library of computer programs that provides an interface between application programs and system hardware components. It performs process management, interprocess communication and process synchronization, memory management, and input/output (I/O) management. The process management module is responsible for process creation, process loading and execution control, the interaction of the process with signal events, process monitoring, CPU allocation and process termination. Interprocess communication covers issues such as synchronization and coordination, deadlock and livelock detection and handling, process protection, and data exchange mechanisms. Memory management includes services for file creation, deletion, reposition, and protection. I/O management handles request and release subroutines for a variety of peripheral and I/O device read, write, and reposition programs.

16.1.1 Real-time operating systems basics

A real-time operating system or RTOS (sometimes known as a real-time executive, real-time nucleus or kernel) is an operating system that implements rules and policies concerning time-critical allocation of system resources. An ordinary operating system is normally designed to provide logical correctness only, but a real-time operating system involves both logical and timewise correctness.

Real-time industrial control systems are classified as hard, firm or soft systems, as discussed in chapter 1, and real-time operating systems need to be capable of satisfying the needs of each. In hard real-time control systems, failure to meet response-time constraints leads to system failure. Firm real-time systems have hard deadlines, but where a certain, low probability of missing a deadline can be tolerated. Systems in which performance is degraded, but not destroyed, by failure to meet response-time constraints are called soft real-time systems.

A RTOS will typically use specialized scheduling algorithms to produce deterministic behavior. It is valued more for how quickly and or predictably it can respond to a particular event than for how much work it can perform in a given time. The key factors in a RTOS are therefore minimal interrupt response time, fixed-time task (also called process) switching, a minimal task scheduling latency, and dynamic memory allocation, discussed in the following paragraphs.

(1) Fixed-time task switching

An ordinary (non-preemptive) operating system might do task switching only at timer tick times, which might, for example, be ten milliseconds apart. In more sophisticated task switching, the

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10016-6

scheduler in an operating system may search through arrays of tasks to determine which should be run next, thus making them non-deterministic. Real-time operating systems, on the other hand, avoid such searches by using incrementally updated tables that allow the task scheduler to identify the task that should run next. In this way the RTOS achieves fixed-time task switching.

Preemptive task switching or task preemption is the act of temporarily interrupting a task, without requiring its cooperation, and with the intention of resuming it at a later time. This is known as a context switch, normally carried out by a privileged task or part of the system known as a preemptive scheduler. A preemptive scheduler has the power to preempt or interrupt, and later resume, system tasks.

There are two types of timing behavior for task switching, as can be seen in Figure 16.1. In a standard operating system, the task-switching time generally rises as the number of tasks to schedule increases. However, the actual time for a task switch is not the time shown by the dashed line, but may be well above or well below it. The shaded regions in the figure show how the actual task-switching time follows what is predicted. The horizontal solid line shows how the task-switching time of a real-time operating system compares with this. It is constant; independent of any load factor.

In some instances, such as the leftmost area of Figure 16.1, the task-switching time may be quicker for a standard operating system than for an RTOS, but for real-time embedded applications this is not important. In fact, the term real-time does not mean as fast as possible but rather indicates consistency and repeatability; that is, it is deterministic.

To have fixed-time task switching, a RTOS needs to have information about the timing performance of its final system. The behavior metrics to be specified are:

(a) Task-switching latency: task-switching or context-switching latency is the time needed to save the context of a currently executing task and switch into another task. It is important that this is short enough.

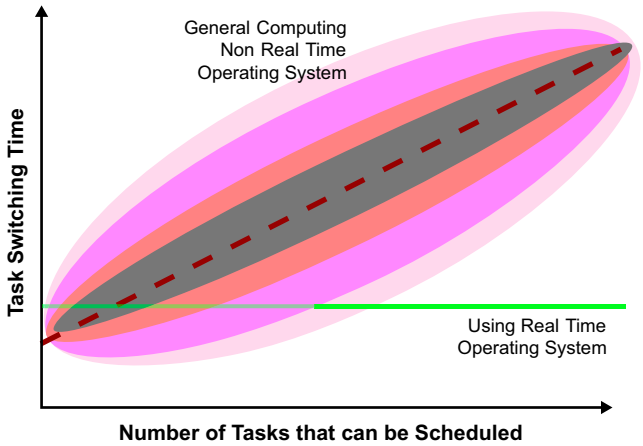


FIGURE 16.1

Task-switching timing in common and real-time operating systems.

(b) Interrupt latency: this is the time that elapses between the execution of the last instruction of the interrupted task and the first instruction in the interrupt handler, or simply the time from interrupt to task run. This is a metric of the response of the system to an external event.
(c) Interrupt dispatch latency: this is the time to go from the last instruction in the interrupt handler to the task next scheduled to run, i.e. the time needed to go from interrupt level to task level.

(2) Task scheduling and synchronization

Most operating systems offer a variety of mechanisms for scheduling and synchronization, necessary in a preemptive environment containing many tasks, because without them the tasks might be well corrupted. They are also provided by a RTOS, by means of the following implementations.

(a) Multitasking and preemptibility

The scheduler in a RTOS should be able to preemp any task in the system, and give the resource to the task that needs it most. It should also be able to handle multiple levels of interrupts; the RTOS should not only be preemptible at task level, but at the interrupt level as well.

(b) Task priority and priority inheritance

In order to preempt tasks effectively, a RTOS needs to be able to determine which task has the earliest deadline to meet, so each task is assigned a priority level. Deadline information is converted to priority levels which are used by the operating system to allocate resources. When using priority scheduling, it is important that the RTOS has a sufficient number of possible levels, so that applications with stringent priority requirements can be implemented. Unbounded priority inversion occurs when a higher-priority task must wait for a low-priority task to release a resource at the same time as the low-priority task is waiting for a medium-priority task. The RTOS can prevent this by giving the lower-priority task the same priority as the one that is being blocked (called priority inheritance). In this case, the blocking task can finish execution without being preempted by a medium-priority task.

(c) Predictable task synchronization

For multiple tasks to communicate with each other in a timely fashion, predictable intertask communication and synchronization mechanisms are required. The ability to lock and unlock resources should be supported, as should the ability to access critical sections whilst maintaining data integrity. For hard real-time industrial control systems, mechanisms and policies to ensure consistency and minimize worst-case blocking, without incurring unbounded or excessive run-time overheads are required. Since most recent work in maintaining the integrity of shared data has been carried out in the context of database systems, these control techniques could be adpated to RTOS task scheduling. The techniques adapted must employ semantic information that is necessarily available at design time to guarantee optimum scheduling.

(3) Determinism and high-speed message passing

Multitasking systems must share data and hardware resources among multiple tasks, because it is usually unsafe for more than two tasks to access the same data or hardware resource simultaneously. This is a critical area of a computer or a control system, which must be protected from access collisions among the tasks that are accessing them. There are four approaches to resolve this

problem: temporarily masking and disabling interrupts; binary semaphores; mutexes for mutual exclusion; and message passing.

Intertask (or interprocess) message communication is an area where different operating systems have different characteristics. Most actually copy messages twice during the transfer process. An approach that avoids this non-determinism, and also accelerates performance is to copy a pointer (a program pointer takes a register in memory) to the message, and deliver that pointer to the message-receiver task without moving the message contents at all. In order to avoid access collisions, the operating system then needs to go back to the message-sender task and delete its copy of the pointer. For large messages, this eliminates the need for lengthy copying and eliminates non-determinism.

(4) Dynamic memory allocation

Determinism of service times is also an issue in the area of dynamic memory allocation. Many standard operating systems offer memory allocation services from what is termed the heap. Each task can temporarily borrow some memory heap from the operating system and specify the size of memory buffer needed. When this task (or another task) has finished with this memory buffer it returns to the operating system which returns it to the heap, where its memory can be used again.

Heaps suffer from a phenomenon called memory fragmentation, which may cause its services to degrade. This fragmentation is caused by the continued reuse of the buffer, sometimes in smaller pieces. When a buffer is no longer needed, it is released and frees the memory. After a heap has undergone many cycles of "in use" and "free", small slivers of memory may appear between memory buffers that are being used by tasks. Over time, a heap will have more and more of these slivers, which will eventually result in situations where tasks requesting memory buffers of a certain size will be refused by the operating system, even though there is enough available memory in its heap. This problem can be solved by so-called garbage collection (defragmentation) software, but algorithms used are often wildly non-deterministic, injecting randomly appearing random-duration delays into heap services.

Real-time operating systems solve this problem by avoiding altogether both memory fragmentation, garbage collection, and their consequences. A RTOS can offer non-fragmenting memory allocation techniques instead of heaps by limiting the variety of memory chunk sizes they make available to application software. While this approach is less flexible than memory heaps, it does avoid memory fragmentation and the need for defragmentation that these generate. For example, the pools memory allocation mechanism allows application software to allocate chunks of memory of perhaps four or eight different buffer sizes per pool. Pools totally avoid external memory fragmentation, by not permitting a buffer that is returned to the pool to be broken into smaller buffers in the future. Instead, the buffer is put onto a free buffer list of buffers of its own size that are available for future re-use at this same size. This is shown in Figure 16.2.

(5) Minimizing interrupt response time

The most important characteristic of a RTOS is its ability to service interrupts quickly. A failure to meet an interrupt response time requirement in a real-time control system can be catastrophic, thus a RTOS must ensure that interrupt-handling time is minimized. If possible, the interrupt handler of an operating system should save the context, create a task that will handle the interrupt service, and return control back to the operating system. When the system fires an interrupt, its microprocessor (or CPU) executes an interrupt service routine (ISR). The amount of time that elapses between a device interrupt

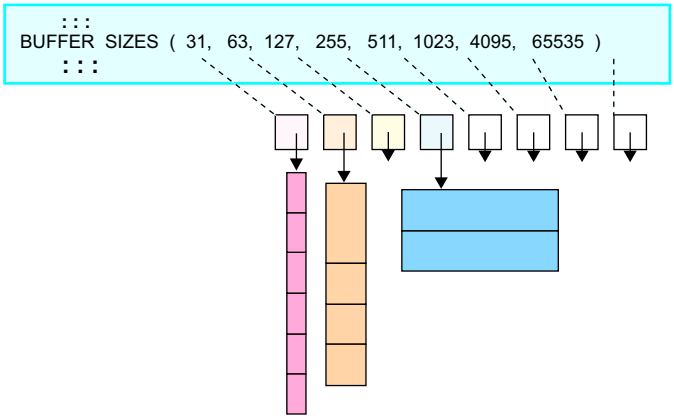


FIGURE 16.2

A memory pool's free buffer lists.

request and the first instruction of the corresponding ISR is known as interrupt latency. The ISR can optionally execute an operating system instruction that will cause a task to be awakened. The amount of time that elapses between the interrupt request and the first instruction of the task that is awakened to handle it is known as task response time. In order to compute the system's worst-case response time, it is necessary to examine all of the sources of interrupt response delays, to ascertain which causes the longest delay to the servicing of the highest-priority interrupt.

Theoretical worst-case delays in interrupt latency and task response time may depend on the choice of CPU, choice of operating system, and how device drivers and other software programs are written. Simple rules and sound programming techniques coupled with proper RTOS interrupt architecture can ensure minimum response times. There are five methodologies that can be used to avoid worst-case delays in both interrupt latency and task response time:

(a) keep all of ISR code as simple and short as possible;
(b) do not disable interrupts;
(c) avoid instructions that increase latency;
(d) avoid improper use of operating system calls in ISR;
(e) properly prioritize interrupts relative to tasks.

A RTOS operates on a set of program structures commonly defined as classes, which is especially true in object-oriented software. Each class in an operating system supports a set of operators, commonly called kernel services, that are invoked by application processes or external events to achieve an expected behavior. For most industrial applications, a RTOS should include these program classes:

(a) Tasks that manage execution of program code; each task is independent of other tasks but can establish many forms of relationships with other tasks in, including data structures, input, output, or other constructs.
(b) Intertask communications, which are mechanisms to pass information from one task to another. Commonly used classes for intertask communications include semaphores, messages and mailboxes, queues, pipes and event flags.

(c) Semaphores, which provide a means of synchronizing tasks with events.
(d) Mutexes, which permit a task to gain exclusive access to an entity or resource.
(e) Timers and alarms, which count ticks and generate signals.
(f) Memory partitions, which manage RAM to prevent fragmentation.
(g) Queues (or pipes, or mailboxes), which permit the passing of fixed amounts of data from a producer to a consumer in FIFO order.
(h) Messages, which are useful in managing variable size data transmissions from a sender to a receiver.
(i) Kernel services, which are routines that are executed to achieve a certain behavior of the system. When an application code entity requires a function provided by the kernel, it initiates a kernel service request for that function.
(j) Event broker or handler, which is a programming paradigm in which the flow of the program is determined by sensor outputs or user actions (mouse clicks, key presses) or messages from other programs or threads. In embedded systems the same may be achieved using interrupts instead of a constantly running main loop; in that case the former portion of the architecture resides completely in hardware.
(k) ISR (interrupt service routine), which is a software routine that is activated to respond to an interrupt.

16.1.2 Real-time operating systems for different platforms

A real-time system is called an embedded control system when the software is encapsulated by the hardware it controls. The microprocessor chipset used to control the mixture of fuel with air in the carburettor of many automobiles is an example of a real-time embedded control system. A RTOS differs from a standard operating system in that the a user of the former can access the microprocessor and peripherals directly. Such an ability helps to meet deadlines.

Any embedded control system has a set of hardware components on which processing can operate. In an embedded control system, they constitute the system with which the operating system is working. Any RTOS also requires a compatible hardware platform. There are two of basic assumptions made about the hardware platforms, as specified below.

(1) Real-time operating systems for single-microprocessor platforms

The assumptions below are the commitments made by the single-microprocessor platforms to meet the requirements of real-time operating systems:

(a) The microprocessor should provide sufficient processing throughput to meet the time requirements of each application.
(b) The microprocessor should have access to or provide the required I/O devices and memory resources.
(c) The microprocessor should have access to timer resources to enable sharing of CPU cycles based on system time.
(d) The microprocessor should provide a mechanism for the RTOS to take control if an application attempts to perform an operation that is not valid. Valid operations are internal to an application, or they may cross the boundary between an application and the modules or components with

which they interact. If they cross that boundary, the interactions should be identified, agreed, and verified. They form a set of commitments that the application supplier must convey to the embedded control system integrator, platform supplier, and RTOS supplier.

(2) Real-time operating systems for multiprocessor platforms

Embedded multiprocessor systems typically have a microprocessor controlling each device. Most real-time operating systems that are multiprocessor-capable use a separate instance of the kernel on each microprocessor. The multiprocessor capability comes from the kernel's ability to send and receive information between microprocessors. In many systems that support multiprocessing there is no difference between the single-microprocessor case and the multiprocessor case from the task's point of view. The RTOS uses a table in each local kernel that contains the location of each task in the system. When one task sends a message to another, the local kernel looks up the location of the destination task and routes the message appropriately. From the task's point of view, all tasks are executing on the same microprocessor.

Multiprocessors can be configured into a variety of forms, from loosely coupled computing grids that use the Internet for communication, to tightly coupled shared-resource architectures referred to as symmetric multiprocessors, or SMPs. With the latter all microprocessors (generally configured into pairs or groups of four) are identical and share a set of resources. This model lends itself to certain programming approaches, and RTOS facilities that can be exploited to benefit the developer. Since all microprocessors have access to the same physical memory, a process or a task can run on any microprocessor from the same memory location. This is the key to adapting an application to SMP architecture.

To utilize the resources of a SMP system, the RTOS must meet the following requirements:

(a) The RTOS that runs on SMP architecture must be ported to work with the SMP and the underlying microprocessor architecture.
(b) The RTOS enables utilization of all the microprocessors.
(c) The RTOS must get all the microprocessors to work on the data simultaneously.
(d) The RTOS must manage operation of multiple instruction streams on independent microprocessors.
(e) The RTOS must provide a mechanism to handle inter-processor (note: not "interprocess") communication.
(f) The RTOS must enable synchronization among the microprocessors.

One method of achieving programming transparency in a multiprocessor system is to assign individual tasks to run on specific microprocessors based on their availability. In this way, the processing load can be shared among microprocessors, with work automatically assigned to one that is free. The RTOS must determine whether a microprocessor is free and, if it is, then a task can be run on it even though the others may already be running other tasks.

Priorities are important to consider, because the RTOS scheduler is designed to maintain priority execution of all tasks. Tasks can therefore safely assume that no lower-priority task can also be executing concurrently. The RTOS must preserve this rule even in the case of a SMP, or the underlying logic upon which an application might be based could falter, and hence may not perform as intended.

Priority-based, preemptive scheduling uses a multiple core of microprocessors to run tasks that are ready. The scheduler automatically runs tasks on available cores. A task that is ready can run on

microprocessor-n if and only if it is of the same priority as the task(s) running on microprocessor(s)-(n−1). After initialization, the RTOS scheduler determines the highest-priority task that is ready to run. It sets the context for that task, and runs it on microprocessor-1. The scheduler then determines whether another task of equal priority is also ready. If so, that task is run on microprocessor-2, and so on. If no additional tasks are ready to run, the scheduler goes to an idle state awaiting either an external event, a service request, interrupt causing preemption, task resume, task sleep or relinquish or finally, task exit.

Preemption occurs when a task is made ready to run at the same time as a lower-priority task is already running. In this event, the lower-priority task is suspended (context saved into heaps), the higher-priority task is started (context restored or initialized), as are any lower-priority tasks on other microprocessors. This is critical to maintain the priority order of executing tasks.

Within this automatic load-balancing approach to managing the resources of SMP architecture, additional features are beneficial to overall performance:

(a) One microprocessor can be made responsible for all external interrupt handling (does not include inter-processor interrupts needed for synchronization or communication).
(b) This leaves the other microprocessor(s) with virtually zero overhead from interrupt handling, enabling it (them) to focus all of its (their) cycles on application processing, even during periods of intense interrupt activity that otherwise might degrade performance.

In the example given in Figure 16.3, we consider a system with two microprocessors, intended to handle a continuous stream of incoming data, such as streaming video. The data must be decompressed

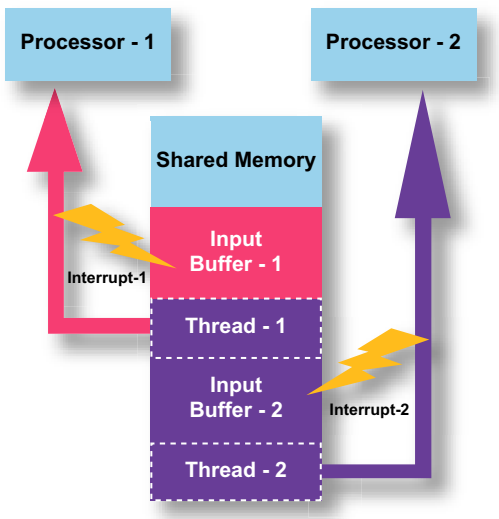


FIGURE 16.3

A typical data flow and processing model using a RTOS with automatic load-balancing support for a SMP of two microprocessors.

in real-time. This is a typical data flow and processing model using a RTOS with automatic load-balancing support for an SMP.

In this example, the input is set up to fill Buffer-1 in memory, with an interrupt generated upon a buffer-full condition (or based on input of a specified number of bytes). As Buffer-1 reaches a full condition:

(a) Buffer-1 FULL generates Interrupt-1;
(b) the ISR handling Interrupt-1 marks Task-1 READY-TO-RUN;
(c) the scheduler runs Task-1 on Microprocessor-1;
(d) data are directed to Buffer-2;
(e) Microprocessor-2 remains idle.

Then, more data arrive while Task-1 is still active, and Buffer-2 fills up:

(f) Buffer-2 FULL generates Interrupt-2;
(g) the ISR handling Interrupt-2 marks Task-2 READY-TO-RUN;
(h) the scheduler runs Task-2 on Microprocessor-2.

16.2 TASK CONTROLS

16.2.1 Multitasking concepts

Any task, or process, running on a computer control system requires certain resources to be able to run. The simplest ones merely have a requirement for some time in the CPU and a volume of system memory, while more complex processes may need extra resources (a serial port, perhaps, or a certain file on a hard disk, or a tape driver). The basic concept of multitasking, often called multi-programming, is to allow the operating system to split the time over which each system resource is available into small chunks, so allowing each of the running processes in turn to access its desired resources. Different processes have different requirements, so the essence of a multitasking system is in the cleverness of the algorithm that allocates resources. As microprocessors have become more powerful, so it has become possible to implement increasingly powerful and complex resource allocation algorithms without compromising the speed of the computer itself.

Most mainstream computers probably have a CPU with a single-core microprocessor. However, it can still do more than one task at the same time with very little, if any, drop in speed seeming to be processing two sets of code at the same time. A typical single-core microprocessor cannot process more than one line of code at any given instant, so it is actually switching quickly from one task to the next, creating an illusion of simultaneous processing. On computers with more than one CPU (called multiprocessor machines), multitasking allows many more tasks to be run than there are CPUs by using two basic methods of cooperative multitasking and preemptive multitasking.

(1) Cooperative multitasking

When one task is already occupying the microprocessor, a wait line is formed for others that also need to use the CPU. Each application is programmed such that after a certain number of cycles, the program will step down and allow other tasks to take their processor time. However, this cooperation schema is rather outdated, and is hampered by limitations. Programmers were free to decide whether

and when their application would surrender CPU time. In the perfect world, every program would respect the other programs running alongside it, but unfortunately, this is not the case. Even when program A could be using CPU cycles while program B and program C are waiting in line, there is no way to stop program A unless it voluntarily steps down.

(2) Preemptive multitasking

The inefficiency of cooperative multitasking left the computer industry scrambling for different ideas, resulting in a new standard called preemptive multitasking, in which the system has the power to halt or "preempt" one code from hogging CPU time. After forcing an interrupt, the operating system has control and can appropriately hand CPU cycle time to another task. Inconvenient interrupt timing is the greatest drawback of preemptive multitasking, but in the end, it is better that all programs see some CPU time rather than having a single program work very slightly faster. Preemptive and cooperative multitasking are, as mentioned, "illusion" multitasking. There are microprocessors that can physically address two streams of data simultaneously; dual or multiprocessor, dual or multicore, and simultaneous multithreading.

The main problem that multitasking introduces in a system is deadlock, which occurs when two or more processes are unable to continue running because each holds some resources that the other needs. Imagine you have processes A and B and resources X and Y. Each process requires both X and Y to run, but process A is holding onto resource X, and B is holding onto Y. Clearly, neither process can continue until it can access the missing resources so the system is said to be in deadlock.

Fortunately, there are a number of ways of avoiding this problem. Option one is to force each process to request all of the resources it is likely to need to run before it actually begins processing. If all resources are not available, it has to wait and try again later. This works but is inefficient (a process may be allocated a resource but not use it for many seconds or minutes, during which time the resource is unavailable). Alternatively, we could define an ordering for resources, such that processes are required to ask for them in the order stipulated. In our example above, we could say that resources must be requested in alphabetical order. So processes A and B would both have to ask for resource X before asking for Y; we could never reach the situation where process B is holding resource Y, because it would have had to be granted access to resource X first.

16.2.2 Task properties

When an operating system is running, each of its tasks is an executing object. To control them at run-time, an operating system is designed to specify every task by their respective properties at any run-time instance. These task properties are primarily task types, task stack and heap, task states, and task body.

(1) Task types

A task type is a limited type that mainly depends on the task attributes given in Table 16.1. Hence, neither assignment nor the predefined comparison for equality and inequality are defined for objects of task types; moreover, the mode out is not allowed for a formal parameter whose type is a task type.

A task object is an object whose type is a task type. The value of a task object designates a task that has the entries of the corresponding task type, whose execution is specified by the corresponding task

Table 16.1 Some Task Attributes

Name	Description
TaskID	Specifies a string that identifies the task. This attribute is always required and must be unique.
ParentTaskID	If this task is to be nested under a previously defined task in the tasks panel, this value contains the TaskID of the parent task.
ResourceBundle	Specifies the dotted notation name for the resource bundle that should be used to retrieve all translatable strings used for items such as menu labels.
Title	Specifies the key of the resource to retrieve from the resource bundle specified as ResourceBundle. This attribute is used as the label under the tasks icon in the Tasks panel. If no resource bundle was specified or if no resource exists in the bundle associated with this key value, this value is used as the icon's label text. If this attribute is not specified, the task's icon is not to be displayed in the Tasks panel.
LiteralTitle	Specifies whether the Title property value is a literal or a key in the ResourceBundle (default). To make the Task title a literal value, use LiteralTitle true. This attribute should only be used for tasks that are created dynamically from end user input.
TaskUnrestricted	If the task should be accessible by all users regardless of how the permissions are set, set this value to true.

body. If a task object is the object, or a subcomponent of the object, declared by an object declaration, then the value of the task object is defined by the elaboration of the object declaration. If a task object is the object, or a subcomponent of the object, created by the evaluation of a scheduler, then the value of the task object is defined by the evaluation of the scheduler. For all parameter modes, if an actual parameter designates a task, the associated formal parameter designates the same task; the same holds for a subcomponent of an actual parameter and the corresponding subcomponent of the associated formal parameter; finally, the same holds for generic parameters.

(2) Task stack and heap

The process or task stack is typically an area of prereserved main storage (system memory) that is used for return addresses, procedure arguments, temporarily saved registers, and locally allocated variables. The microprocessor typically contains a register that points to the top of the stack, called the stack pointer. It is implicitly used by machine code instructions that call a procedure, return from a procedure, store or fetch a data item to/from the stack.

The process or task heap is also an area of prereserved main storage (system memory) that a program process can use to store data in some variable amount that is unknown until the program is running. For example, a program may accept different amounts of input from one or more users for processing, and then process all the input data at once. Having a certain amount of heap storage already obtained from the operating system makes it easier for the process to manage storage, and is generally faster than asking the operating system for storage every time it is needed. The process manages its allocated heap by requesting a chunk of it (called a heap block) when needed, returning the blocks when no longer needed, and doing occasional garbage collecting, which makes unused blocks available, and also reorganizes the available space in the heap so that it is not being wasted in small, unused pieces.

A stack and a heap are similar to each other, except that the blocks are taken out of storage in a certain order and returned in the same way. The following are some descriptions for the working procedure of the stack, heap, and frame-stack that are used in the execution of a process or task.

(a) Process stack

A stack holds the values used and computed during run time. When the machine calls a function, the parameters are pushed on the top of the stack though they are actually merged to form a single list datum. If the function requires any local variable, the machine allocates space for their values. When the function returns, the allocated local variables and the arguments stored on the stack are popped; and the value is returned by pushing it onto the stack. Figure 16.4(a) shows the working procedure of the process stack; at points before a function is called, during a function call, the arguments having been pushed onto the stack, and after the function has returned, the arguments and local variables having been popped, and the answer pushed onto the stack. Since the stack is a fixed-size memory block, a deep function call may cause a stack overflow error.

(b) Process heap

To reduce the frequency of copying the contents of such data structures as strings and lists, another data structure, called the heap, is used to hold the contents of temporary strings or lists during the string and list operations. The heap is a large memory block. Memory is allocated within this block. A pointer keeps track of the top of the heap, in a similar way to the stack pointer of the stack.

The process heap is not affected by the return of the function call. There are normally some "free-heap" instructions that are automatically inserted by the programmer at the points that it thinks safe to free memory. However, if a computation involves long strings or lists, or too deep a function is called, the machine may not have the chance to free the memory and thus causes a heap overflow error. The programmer may have to modify the program to minimize the load on the stack and the heap to avoid memory overflow errors; for instance, an iterative function has less demand in this regard than its recursive equivalent.

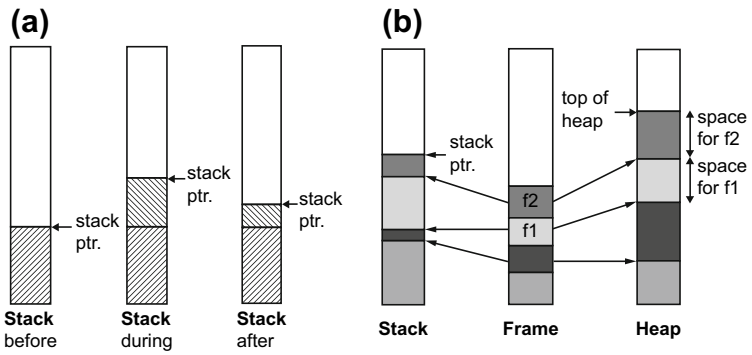


FIGURE 16.4

Process stack, process heap and process frame-stack: (a) the working procedure of the process stack; (b) a comparison between the working procedures between the process stack, process heap and process frame-stack.

(c) Process frame-stack

When returning from a function call, the machine has to store the previous status, for example, stack pointer (pseudo machine) program counter, number of local variables (since local variables were put on the stack), and so on. A frame-stack is a stack that holds this information.

When a function is called, the current machine status is pushed on the frame-stack. For example, if function f1() calls function f2(), the status of stack, heap, and frame-stack is as shown in Figure 16.4 (b). The frame-stack keeps track of the stack and heap pointers. A free-heap instruction causes the top-of-heap to return to the bottom pointed to by the top-most frame-stack (e.g., f2()). On the other hand, the stack pointer is lowered only if the function returns.

A frame-stack is popped when a function returns and the machine status resumes its previous state. Since the frame-stack is implemented as a small stack, say about 64 entries, the number of levels of nested function calls is about 64 in this case. This maximum limit is enough for general use, but if a function is nested too deep, the machine will generate a "nested too deep" error.

(3) Task states

When an application program is executing, at any moment a task is characterized by its state. Normally, the following five task states are defined for tasks in a RTOS:

(a) Running. In the running state, the CPU (or at least one if it is in multiprocessor chipset) is assigned to the task, so that its code instructions can be executed. Only one task can be in this state at any point in time, while all the other states can be adopted simultaneously by several tasks.
(b) Ready. All functional prerequisites for a transition into the running state exist, and the task only waits for allocation of the microprocessor. The scheduler decides which ready task is executed next.
(c) Waiting. A task cannot continue execution because it has to wait for at least one event. Only extended tasks can jump into this state (because they are the only ones that can use events).
(d) Suspended. In the suspended state the task is passive and can be activated.
(e) Terminated. In this task state, the task allocator deletes the corresponding task object and releases its resources.

Figure 16.5 is a diagram of a possible design for the transitions between these five task states. Note that basic tasks (also called main or background tasks) have no waiting state: a basic task can only represent a synchronization point at its beginning and end. Application parts with internal synchronization points have to be implemented by more than one basic task. An advantage of extended tasks is that they can handle a coherent job in a single task, no matter which synchronization requests are active. Whenever current information for further processing is missing, the extended task switches over into the waiting state. It exits this state whenever corresponding events signal the receipt or the update of the desired data or events.

Depending on the conformance class a basic task can be activated once or many times. The latter means that activation issued when a task is not in the suspended state will be recorded and then executed when the task finishes the current instance.

The termination of a task instance only happens when it terminates itself (to simplify the RTOS, no explicit task kill primitives are provided).

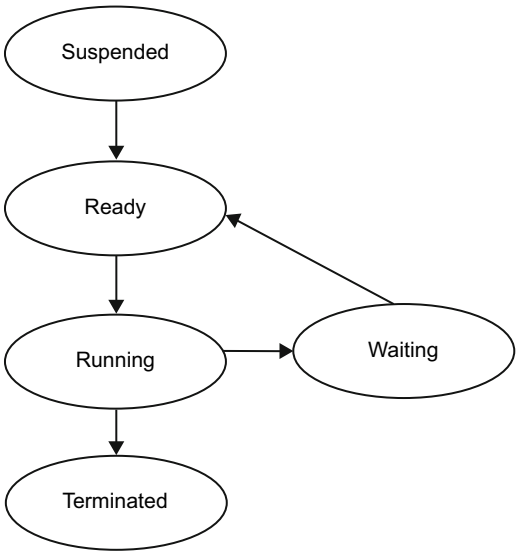


FIGURE 16.5

A possible design for the task state transitions.

(4) Task body

The execution of the task is defined by the corresponding task body. A task body documents all the executable contents of code. Task objects and types can be declared in any declarative part, including task bodies themselves. For any task type, the specification and body must be declared together in the same unit, with the body usually being placed at the end of the declarative part.

The simple name at the start of a task body must repeat the task unit identifier. Similarly, if a simple name appears at the end of the task specification or body, it must also repeat this. Within a task body, the name of the corresponding task unit can also be used to refer to the task object that designates the task currently executing the body; furthermore, the use of this name as a type mark is not allowed within the task unit itself.

The elaboration of a task body has no other effect than to establish that the body can, from then on, be used for the execution of tasks designated by objects of the corresponding task type. The execution of a task body is invoked by the activation of a task object of the corresponding type. The optional exception handlers at the end of a task body handle exceptions raised during the execution of the sequence of statements of the task body.

16.2.3 Task creation and termination

A task object can be created, either as part of the elaboration of an object declaration that occurs immediately within some declarative region, or as part of the evaluation of an allocator. All tasks created by the elaboration of object declarations of a single declarative region (including

subcomponents of the declared objects) are activated together. Similarly, all tasks created by the evaluation of a single allocator are activated together. The execution of a task object has three main active phases:

1. Activation. The elaboration of the declarative part, if any, of the task body (local variables in the body of the task are created and initialized during activation); the activator identifies the task that created and activated the task.
2. Normal execution. In this phase, the execution of the statements is visible within the body of the task.
3. Finalization. The execution of any finalization code associated with any objects in its declarative part.

The parent task is the task on which a task depends, determined by the following rules: If the task has been declared by means of an object declaration, its parent is the task which declared the task object. If the task has been declared by means of an allocator, its parent is the task that has the corresponding access declaration. When a parent creates a new task, the parent's execution is suspended while it waits for the child to finish activating (either immediately, if the child is created by an allocator, or after the elaboration of the associated declarative part). Once the child has finished its activation, parent and child proceed concurrently. If a task creates another during its activation, then it must also wait for its child to activate before it can begin execution.

The master is the execution of a construct that includes finalization of local objects after it is complete (and after waiting for any local task), but before leaving. Each task depends on one or more masters, as follows. If the task is created by the evaluation of an allocator for a given access type, it depends on each master that includes the elaboration of the declaration of the ultimate ancestor of the given access type. If the task is created by the elaboration of an object declaration, it depends on each master that includes its elaboration. Furthermore, if a task depends on a given master, it is defined as depending on the task that executes the master, and (recursively) on any master of that task.

For the finalization of a master, dependent tasks are first awaited. Then each object whose accessibility level is the same as that of the master is finalized if the object was successfully initialized and still exists. Note that any object whose accessibility level is deeper than that of the master would no longer exist; those objects would have been finalized by some inner master. Thus, after leaving a master, the only objects yet to be finalized are those whose accessibility level is not as deep as that of the master.

16.2.4 Task queue

Task queues are the way to defer work until later, in the kernels of the RTOS. A task queue is a simple data structure; see Figure 16.6, which consists of a singly linked list of "tq object" data structures, each of which contains the address of a task body routine and a pointer (of the prefix *) to some data. The routine will be called when the element on the task queue is processed, and it will be passed a pointer to the data. Anything in the kernel, such as a device driver, can create and use task queues but there are four task queues created and managed by the kernel:

(a) Timer. This queue is used to queue work that will be done as soon after the next system clock tick as is possible. At each clock tick, this queue is checked to see whether it contains any entries and, if it does, the timer queue handler is made active. This is processed, along with all the

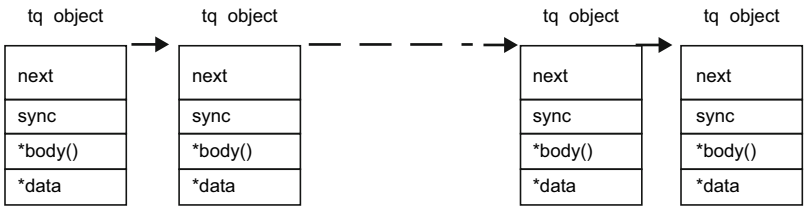


FIGURE 16.6

A task queue.

other handlers, when the scheduler next runs. This queue should not be confused with system timers, which are a much more sophisticated mechanism.

(b) Immediate. This queue is also processed when the scheduler processes the active handlers by means of their priorities. The immediate handler is not as high in priority as the timer queue handler and so these tasks will be run later.

(c) Scheduler. This task queue is processed directly by the scheduler. It is used to support other task queues in the system and, in this case, the task to be run will be a routine that processes a task queue, say, for a device driver or an interface monitor.

(d) Waiting. There are many instances when a process must wait for a system resource before it can carry on. To deal with this, the kernel uses a simple data structure, called a wait queue. When processes are added to the end of a wait queue they can either be interruptible or uninterruptible. Interruptible processes may be interrupted by events such as timers expiring, or signals being delivered whilst they are waiting on a wait queue. Uninterruptible processes can never be interrupted at any time or by any event. When the wait queue is processed, the state of every process in the wait queue is set to "running", and is moved into corresponding task queue. The next time the scheduler runs, the processes that are on the wait queue are now candidates to be run as they are now no longer waiting. When a process on the wait queue is scheduled, the first thing that it will do is remove itself from the wait queue. Wait queues can be used to synchronize access to system resources and they are used by Linux in its implementation of semaphores.

When task queues are processed, the pointer to the first element in the queue is removed and replaced with a null pointer. In fact, this removal is an atomic operation that cannot be interrupted. Then each element in the queue has its handling routine called in turn. The elements in the queue are often statically allocated data; however, there is no inherent mechanism for discarding allocated memory. The task queue processing routine simply moves onto the next element in the list. It is the job of the task itself to ensure that it properly cleans up any allocated kernel memory.

16.2.5 Task context switch and task scheduler

(1) Task context switch

A context switch (also sometimes referred to as a process or a task switch) is the switching of the CPU from one process or task to another. Sometimes, a context switch is described as the kernel

suspending execution of one process on the CPU and resuming execution of some other that had previously been executed. Context switching is an essential feature of multitasking operating systems. The illusion of concurrency is achieved by means of context switches that occur in rapid succession (tens or hundreds of times per second), either as a result of processes voluntarily relinquishing their time in the CPU, or as a result of the scheduler making the switch when a process has used up its CPU time slice.

A context is the contents of a CPU's registers and program counter at any point in time. A register is a small amount of very fast memory inside a CPU (as opposed to the slower RAM main memory outside the CPU) that is used to speed the execution of computer programs by providing quick access to commonly used values, generally those in the middle of a calculation. A program counter is a specialized register that indicates the position of the CPU in its instruction sequence and holds either the address of the instruction being executed, or the address of the next, depending on the specific system. Context switching can be described in slightly more detail as the operating system or kernel performing the following activities:

(a) The state of the first process must be saved so that, when the scheduler returns to it, this state can be restored and execution can continue normally.
(b) The state of the process includes all the registers in use, especially the program counter, plus any other operating-system-specific data that may be necessary. Often, all the necessary data for state are stored in one data structure, called a switchframe or a process control block.
(c) To switch processes, the switchframe for the first process must be created and saved. They are sometimes stored upon a per-process stack in kernel memory (as opposed to the user-mode stack), or there may be some specific operating-system-defined data structure for this information.
(d) Since the operating system has effectively suspended the execution of the first process, it can now load the switchframe and context of the second. In doing so, the program counter from the switchframe is loaded, and thus execution can continue for the new process.

A context switch can also occur as a result of a hardware interrupt, which is a signal from a hardware device to the kernel that an event has occurred. Some processors, such as the Intel 80386 and higher CPUs, have hardware support for context switches, by making use of a special data segment designated the task state segment (TSS). When a task switch occurs (referring to a task gate or explicitly due to an interrupt or exception) the CPU can automatically load the new state from the TSS. Like other tasks performed in hardware, one would expect this to be rather fast; however, mainstream operating systems, including Windows, do not use this feature. This is partly because hardware context switching does not save all the registers (only general-purpose registers, not floating-point registers), and partly due to associated performance issues.

However, most modern operating systems can perform software context switching, which can be used on any CPU, rather than hardware context switching, in an attempt to obtain improved performance. Software context switching was first implemented in Linux for Intel-compatible processors with the 2.4 kernel. One major advantage claimed is that, while the hardware mechanism saves almost all of the CPU state, software can be more selective and save only that portion that actually needs to be saved and reloaded. However, there is some question as to how important this really is in increasing the efficiency of context switching. Its advocates also claim that software context switching allows the switching code to be improved, thereby further enhancing efficiency, and that it permits better control over the validity of the data being loaded.

(2) Task scheduler of common operating systems

For those operating systems without real-time constraints, the process or task scheduler is that part of the operating system which responds to the requests by programs and interrupts for microprocessor attention. A scheduler can also stand alone to act as a centerpiece to a program that requires moderation between many different tasks. In this capacity, each program task is written so that it can be called by the scheduler as necessary. Most embedded programs can be described as having a specific, discrete response to a stimulus or time interval. These tasks can be ranked in priority, allowing the scheduler to hand control of the microprocessor to each process in turn.

The scheduler itself is a loop that calls one of the other processes each time it executes. Each microprocessor executes the scheduler itself and will select its next task from all runnable processes that are not allocated to a microprocessor. An array of flags for the process, then indicates its need for scheduling. The simplest way to handle the problem is to give each program a turn, then when a process gets its turn, can decide when to return control. This method does not support any notion of importance among processes, so it is not as useful as it could be. The more complicated the scheduler or operating system, the more elaborate is the kind of scheduling information that can be maintained.

The scheduling information of a process or task can be measured with the following metrics: (1) CPU utilization, which is the percentage of time that the CPU is doing useful work (i.e., not idling); 100% is perfect; (2) wait time, which is the average time a process spends in the run queue; (3) throughput, which is the number of processes completed per time unit; (4) response time, which is the average time elapsed from the time the process is submitted until useful output is obtained; (5) turnaround time, which is the average time elapsed between when a process is submitted and when it has completed. Typically, utilization and throughput are traded-off for better response time. In general, we would like to optimize the average measure of response time. In some cases, minimum and maximum values are optimized; for example, it might be a good idea to minimize the maximum response time.

The types of process or task schedulers depend on the adopted algorithms, which are generally of the following types:

(a) First-come, first-served (FCFS) . The FCFS scheduler simply executes processes to completion in the order they are submitted. They can use a queue data structure, that, given a group of processes to run, will insert them all into the queue and execute them in that order.

(b) Round-robin (RR) . RR is a preemptive scheduler, which is designed especially for time-sharing systems. In other words, it does not wait for a process to finish or give up control. In RR, each process is given a time slot to run. If the process does not finish, it will get back in line and receive another time slot until it has completed. The implementation of RR can use a FIFO queue where new jobs are inserted at the tail end.

(c) Shortest-job-first (SJF). The SJF scheduler is like FCFS, except that instead of choosing the job at the front of the queue, it will always choose the shortest job (i.e., the job that takes the least time) available. It uses a sorted list to order the processes from longest to shortest, and when adding a new process or task, needs to figure out where in the list to insert it.

(d) Priority scheduling (PS). As mentioned, a priority is associated with each process. This priority can be defined in any way: for example, we can think of the shortest job as top priority, so this algorithm can be a special case of PRI. Processes with equal priorities may be scheduled in accordance with FCFS.

(3) The task scheduler of real-time operating systems

The goals for real-time scheduling are to complete tasks within specific time constraints and to prevent simultaneous access to shared resources and devices (critical sections). Although system resource utilization is of interest, it is not a primary driver, and predictability and temporal correctness are the principal concerns of the task scheduler of real-time operating systems. The algorithms used in real-time scheduling vary from relatively simple to extremely complex. The topic of real-time scheduling algorithms can be studied for either single-processor or multiprocessor platforms.

We first study real-time scheduling algorithms for a single-processor CPU. The set of such algorithms is divided into two major subsets, namely offline and online scheduling algorithms.

Offline algorithms (pre-run-time scheduling) generate scheduling information prior to system execution, which is utilized by the system during run-time. In systems using offline scheduling, there is generally, if not always, a required ordering of the execution of processes. This can be accommodated by using precedence relations that are enforced during offline scheduling. Preventing simultaneous access to shared resources and devices is also necessary and is accomplished by defining which portion of a process cannot be preempted by another task, and then defining exclusion constraints and enforcing them during the offline algorithm. Another goal that may be desired for offline schedules is reducing the cost of context switches caused by preemption.

Offline algorithms are good for applications where all characteristics are known a priori and change very infrequently, since a fairly complete characterization of all processes involved, such as execution times, deadlines, and ready times, is required. The algorithms need large amounts of offline processing time to produce the final schedule, and hence are quite inflexible. Any change to the system processes requires starting the scheduling problem over from the beginning. In addition, they cannot handle an environment that is not completely predictable. However, a major advantage of offline scheduling is a significant reduction in run-time resources, including processing time, used for scheduling.

Online algorithms generate scheduling information while the system is running. These schedulers do not assume any knowledge of process characteristics which have not arrived yet, and so require a large amount of run-time processing time.

However, if different modes or some form of error handling is desired, multiple offline schedules can be computed, one for each alternative situation, so at run-time, a small online scheduler can choose the correct one. One severe problem that can occur with this approach is priority inversion. This occurs when a lower-priority task is using a resource that is required by a higher-priority task and this causes blocking of the higher-priority task by the lower-priority one. The major advantage of online scheduling is that there is no requirement to know tasks' characteristics in advance, and so they tend to be flexible and easily adaptable to environment changes.

Online scheduling algorithms can be static or dynamic priority-based algorithms, which are discussed briefly here.

(a) Online static priority-based algorithms may be either preemptive or non-preemptive. The following two non-preemptive algorithms attempt to provide high microprocessor utilization whilst preserving task deadline guarantees.

Parametric dispatching algorithm: this uses a calendar of functions describing the upper and lower bounds of allowable start times for the task. During an offline component, the timing constraints between tasks are analyzed to generate the calendar of functions, then, during system execution, these

bounds are passed to the dispatcher, which then determines when to start execution of the task. This decision can be based on whether there are other non-real-time tasks waiting to execute.

Predictive algorithm: this algorithm depends upon known a priori task execution and arrival times. When it is time to schedule a task for execution, the scheduler not only looks at the first task in the ready queue, but also looks at the deadlines for tasks that are predicted to arrive prior to the first task's completion. If a later task is expected to arrive with an earlier deadline than the current task, the scheduler may insert CPU idle time and wait for the pending arrival, if this will produce a better schedule. In particular, the insertion of idle time may keep the pending task from missing its deadline.

(b) Online dynamic priority-based algorithms require a large amount of resources. There are two subsets of dynamic algorithms: planning-based and best effort. They attempt to provide better response to aperiodic or soft tasks, while still meeting the timing constraints of the hard, periodic tasks. This is often accomplished by utilization of spare microprocessor capacity to service soft and aperiodic tasks.

Planning-based algorithms reject task execution if they cannot guarantee its on-time completion. Most of them can provide higher microprocessor utilization than static priority-based algorithms while still guaranteeing on-time completion of accepted tasks. Earliest deadline first scheduling is one of the first planning-based algorithms proposed, providing the basis for many of those currently being studied and used. The dynamic priority exchange server, dynamic sporadic server, total bandwidth server, earliest deadline late server, and improved priority exchange server are all examples of planning-based algorithms.

Best effort algorithms that can be used by an application task, are based on application-specified benefit functions such as the energy consumption function. There exist many best effort real-time scheduling algorithms. Two of the most prominent are the dependent activity scheduling algorithm (DASA) and the lock best effort scheduling algorithm (LBESA).

DASA and LBESA are equivalent to the earliest deadline first algorithm (EDF) during under-loaded conditions, where EDF is optimal and guarantees that all deadlines are always satisfied. In the event of an overload situation, DASA and LBESA seek to maximize the aggregate task benefit. The DASA algorithm makes scheduling decisions using the concept of benefit densities. The benefit density of a task is the benefit accrued per unit time by the execution of the task. The objective of DASA is to compute a schedule that will maximize the aggregate task benefit, which is the cumulative sum of the benefit accrued by the execution of the tasks. Thus, since task benefit functions are step-benefit functions, a schedule that satisfies all deadlines of all tasks will yield the maximum aggregate benefit. LBESA is another best effort real-time scheduling algorithm, similar to DASA in that both algorithms schedule tasks using the notion of benefit densities and are equivalent to EDF during under-load situations. However, the algorithms differ in the way they reject tasks during overload situations. While DASA examines tasks in the ready queue in decreasing order of their benefit densities for determining feasibility, LBESA examines tasks in increasing order of deadlines. Like DASA, LBESA also inserts each task into a tentative schedule at its deadline-position and checks the feasibility of that schedule. Tasks are maintained in increasing deadline order in the tentative schedule. If the insertion of a task into it results in an infeasible schedule, then, unlike DASA, LBESA removes the task with least benefit density from the schedule. LBESA does this continuously until the schedule becomes feasible. Once all tasks in the ready queue have been examined and a feasible tentative schedule is thus constructed, LBESA selects the earliest deadline task from it.

The scheduling of real-time systems has been much studied on machines in which there is exactly one shared microprocessor available, and all the processes in the system are required to execute on it.

In multiprocessor platforms there are several microprocessors available upon which these processes may execute.

The following assumptions may be made to design a multiprocessor scheduling algorithm. (1) Task pre-emption is permitted. That is, a task executing on a microprocessor may be preempted before completion, and its execution resumed later. It may be assumed that there is no penalty associated with such pre-emption. (2) Task migration is permitted. That is, a task that has been preempted on a particular processor may resume execution on a different one. Once again, we may assume that there is no penalty associated with such migration. (3) Task parallelism is forbidden. That is, each task may execute on at most one microprocessor at any given instant in time. Multiprocessor scheduling techniques fall into the two general categories, as given below.

(a) Global scheduling algorithms store the tasks that have arrived, but not finished their execution, in one queue which is shared among all microprocessors. Suppose there are k microprocessors in the system. At every moment the k highest-priority tasks of the queue are selected for execution using pre-emption and migration if necessary. The focused addressing and bidding algorithm is an example of a global scheduling algorithm, which works as follows. Each microprocessor maintains a status table that indicates the tasks already committed to run. It also maintains a table of the surplus computational capacity of every other microprocessor in the system. The time axis is divided into windows of fixed duration, and each microprocessor regularly sends to its colleagues the fraction of the next window that is currently free. On the other hand, an overloaded microprocessor checks its surplus information and selects a microprocessor that seems to be most likely to be able to successfully execute that task by its deadline. It then ships the tasks out to that microprocessor, which is called the selected microprocessor. However, the surplus information may be outdated, and it is possible that the selected microprocessor will not have the free time to execute the task. In order to avoid this problem, and in parallel with sending out the task to the selected processor, the originating microprocessor asks other lightly loaded microprocessors how quickly they can successfully process the task. The replies are sent to the selected microprocessor, which, if unable to process the task successfully, can review the replies to find one able to do so, and transfers the task to that microprocessor.

(b) Partitioning scheduling algorithms divide the set of tasks such that all tasks in a partition are assigned to the same microprocessor. Tasks are not allowed to migrate hence the multiprocessor scheduling problem is transformed into many single-processor scheduling problems. The next fit algorithm is a multiprocessor scheduling algorithm that works on the basis of a partitioning strategy. A set of classes of the task are defined. Tasks are allocated one by one to the appropriate microprocessor class, then, with this assignment, the scheduling algorithm is running on each microprocessor.

In addition to the above approaches, we can apply hybrid partitioning and global strategies. For instance, each process can be assigned to a single microprocessor, while a task is allowed to migrate.

16.2.6 Task threads

A thread is a single sequential flow of control within a program. Task threads are a way for a program to split a task into two or more simultaneously running subtasks. Multiple threads can be executed in parallel on many control systems. This multithreading generally occurs by time slicing (where a single microprocessor switches between different threads) or by multiprocessing (where threads are executed on separate microprocessors).

A single thread also has a beginning, a sequence, and an end. At any given time during the runtime of the thread, there is a single point of execution. However, a thread itself is not a program; it cannot run on its own. Rather, a thread runs within a program, as shown in Figure 16.7(a). The real potential for threads is not about a single one, but about the use of multiple threads running at the same time and performing different tasks in a single program. Figure 16.7(b) shows this multithreading in a program.

Threads are distinguished from traditional multitasking operating system processes in that processes are typically independent, carry considerable state information, have separate address spaces, and interact only through system-provided interprocess communication mechanisms. Multiple threads, on the other hand, typically share the state information of a single process, and share memory and other resources directly. Context switching between threads in the same process is typically faster than context switching between processes.

One advantage of a multithreaded program is that it can operate faster on multiprocessor CPUs that are CPUs with multiple cores, or across a cluster of machines, because the threads of the program naturally lend themselves to truly concurrent execution. In such a case, the programmer needs to be careful to avoid race conditions and other non-intuitive behaviors by implementing correct scheduler algorithms. In order for data to be correctly manipulated, threads will often need to rendezvous in time to process data in the correct order. Threads may also require atomic operations (often implemented using semaphores) to prevent common data from being simultaneously modified, or read while in the process of being modified. Careless use of such primitives can lead to deadlocks.

Many modern operating systems support both time-sliced and multiprocessor threading directly with a process scheduler. Some implementations are called kernel threads or lightweight processes. In the absence of such mechanisms, programs can still implement threading by using timers, signals, or other methods to interrupt their own execution and hence perform a sort of ad hoc time-slicing. These are sometimes called user-space threads. Both standard and real-time operating systems generally implement threads, either by preemptive multithreading or cooperative multithreading. Preemptive multithreading is the same as preemptive tasking, and is generally considered the superior implementation, as it allows the operating system to determine when a context switch should occur.

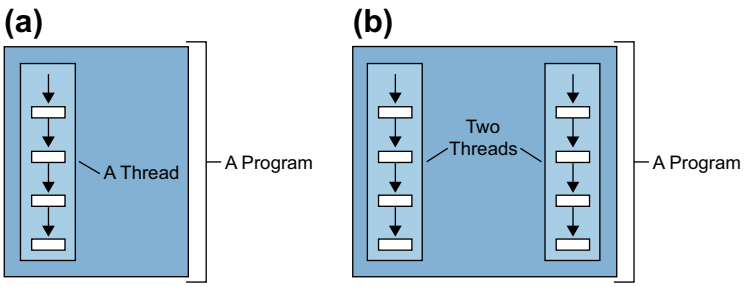


FIGURE 16.7

A task thread illustration: (a) the relationship of the task program to its task threads; (b) two threads running concurrently in a single program.

Cooperative multithreading, on the other hand, relies on the threads themselves to relinquish control once they are at a stopping point, which is also the same as cooperative tasking. This can create problems if a thread is waiting for a resource to become available. The disadvantage to preemptive multithreading is that the system may make a context switch at an inappropriate time, causing priority inversion or other bad effects that may be avoided by cooperative multithreading.

Traditional mainstream computing hardware did not have much support for multithreading, as switching between threads was generally already quicker than full process context switches. Microprocessors in embedded systems, which have higher requirements for real-time behaviors, might support multithreading by decreasing the thread switch time, perhaps by allocating a dedicated register file for each thread instead of saving/restoring a common register file. In the late 1990s, the idea of executing instructions from multiple threads simultaneously became known as simultaneous multithreading. This feature was introduced in Intel's Pentium 4 processor, with the name Hyper-threading.

16.3 INPUT/OUTPUT DEVICE DRIVERS

16.3.1 Input/output devices

Input/output is abbreviated to I/O. The I/O subsystem, composed of I/O devices, device controllers, and associated input/output software, is a major component of both computer and industrial control systems. All I/O devices are classified as either block or character devices. A block special device causes I/O to be buffered in large pieces, whereas the character device causes I/O to occur one character (byte) at a time. Some devices can be both block and character devices, and must have entries for each mode.

Input and output devices allow the computer system to interact with the outside environment. Some examples of input devices are keyboard, mouse, microphone, bar code reader, and graphics tablet, and examples of output devices are monitor, printer, and speaker.

For industrial control, a good example of I/O systems is the data acquisition I/O modules, which are important subsystems in most industrial control systems, in particular in SCADA networks. Data acquisition is the processing of multiple electrical or electronic inputs from devices such as sensors, timers, relays, and solid-state circuits for the purpose of monitoring, analyzing and/or controlling systems and processes. Device performance, analog outputs, form factor, computer bus, connection to host, and environmental parameters are all important specifications to consider for data acquisition I/O modules.

One of the important tasks of an operating system is to control all of its I/O devices, which includes such tasks as issuing commands concerning data transfer or status polling, catching and processing interrupts, as well as handling different kind of errors. This is the topic of this section.

16.3.2 Device drivers

Device drivers are specific programs which contain device-dependent codes. Each can handle one device type, or one class of closely related devices. For example, some kinds of dumb terminal can be controlled by a single terminal driver. On the other hand, a dumb hardcopy terminal and an intelligent graphics terminal are so different that different drivers must be used. Each device controller has one or

more registers used to receive its commands. The device drivers issue these commands, and check that they are carried out properly. Thus, a communication driver is the only part of the operating system that knows how many registers the associated serial controller has, and what they are used for. In general, a device driver has to accept requests from the device-independent software above it, and to check that they are carried out correctly. For example, a typical request is to read a block of data from the disk. If the device driver is idle, it starts carrying out the request immediately, but if it is already busy with another request, it will enter the new request into a queue, which will be dealt with as soon as possible.

To carry out an I/O request, the device driver must decide which controller operations are required, and in what sequence. It starts issuing the corresponding commands by writing them into the controller's device registers. In many cases, the device driver must wait until the controller does some work, so it blocks itself until the interrupt comes in to unblock it. Sometimes, however, the I/O operation finishes without delay, so the driver does not need to block. After the operation has been completed, it must check for errors. Status information is then returned back to its caller. Buffering is also an issue, for both block and character devices. For block devices, the hardware generally insists on reading and writing entire blocks at once, but user processes are free to read and write in arbitrary units. If a user process writes half a block, the operating system will normally keep the data internally until the rest of the data are written, at which time the block can go out to the disk. For character devices, users can write data to the system faster than it can be output, necessitating buffering. A keyboard input can also arrive before it is needed, also requiring buffering.

Error handling is done by the device drivers. Most errors are highly device-dependent, so only the device driver knows what to do, such as retry or ignore, and so on. A typical error is caused by a disk block that has been damaged and cannot be read any more. After the driver has tried to read the block a certain number of times, it gives up and informs the device-independent software about the error. How it is treated from here on is a task for the device-independent software. If the error occurred while reading a user file, it may be sufficient to report the error back to the caller. However, if it occurred while reading a critical system data structure, such as the block containing the bit map showing which blocks are free, the operating system may have no choice but to print an error message and terminate.

(1) Device driver content

Device drivers make up the major part of all operating systems kernels. Like other parts of the operating system, they operate in a highly privileged environment and can cause disaster if they get things wrong. Device drivers control the interaction between the operating system and the device that they are controlling. For example, the file system makes use of a general block device interface when writing blocks to an IDE disk. The driver takes care of the details and makes device-specific things happen. Device drivers are specific to the controller chip that they are driving, which is why, for example, you need the NCR810 SCSI driver if your system has an NCR810 SCSI controller.

Every device driver has two important data structures; the device information structure and the static structure. These are used to install the device driver and to share information among the entry point routines. The device information structure is a static file that is passed to the install entry point. The purpose of the information structure is to pass the information required to install a major device into the install entry point where it is used to initialize the static structure. The static structure is used to

pass information between the different entry points, and is initialized with the information stored in the information structure. The operating system communicates with the driver through its entry point routines.

(2) Device Driver status

The entry point routines provide an interface between the operating system and the user applications. For example, when a user makes an open system call, the operating system responds by calling the open entry point routine, if it exists. There is a list of defined entry points, but every driver does not need to use all of them.

(a) Install. This routine is called once for each major device when it is configured into the system. The install routine is responsible for allocating and initializing data structures and the device hardware, if present. It receives the address of a device information structure that holds the parameters for a major device. The install routine for a character driver should follow this pseudocode.

(b) Open. The open entry point performs the initialization for a minor device. Every open system call results in the invocation of the open entry point. The open entry point is not re-entrant; therefore, only one user task can be executing this entry point's code at any time for a particular device.

(c) Close. The close entry point is invoked when the last open file descriptor for a particular device file is closed.

(d) Read. The read entry point copies a certain number of bytes from the device into the user's buffer.

(e) Write. The write entry point copies a certain number of bytes from the user's buffer to the device.

(f) Select. The select entry point supports I/O polling or multiplexing. The code for this entry point is complicated and the discussion of this code or structure is probably better left until needed. Unless you have a slow device, most likely it will never be needed.

(g) Uninstall. The uninstall entry point is called once when the major device is uninstalled from the system. Any dynamically allocated memory or interrupt vectors set in the install entry point should be freed in this entry point.

(h) Strategy. The strategy entry point is valid only for block devices. Instead of having a read and write entry point, block device drivers have a strategy entry point routine that handles both reading and writing.

16.3.3 Request contention

Dealing with race conditions is one of the difficult aspects of an I/O device driver. The most common way of protecting data from concurrent access is I/O request contention, traditionally operating by means of the I/O request queue.

The most important function in a block driver is the request function, which performs the low-level operations related to reading and writing data. Each block driver works with at least one I/O request queue. This queue contains, at any given time, all of the I/O operations that the operating system would like to see done on the driver's devices. The management of this queue is complicated; the performance of the system depends on how it is done. The I/O request queue is a complex data structure that is accessed in many places in the operating system. It is entirely possible that the operating system needs to add more requests to the queue at the same time that the device driver is taking requests off it. The queue is thus subject to the usual sort of race conditions, and must be protected accordingly.

A variant of this latter case can also occur if the request function returns while an I/O request is still active. Many drivers for real hardware will start an I/O operation, then return; the work is completed in the driver's interrupt handler. We will look at interrupt-handling methodology in detail later in this chapter; for now it is worth mentioning, however, that the request function can be called while these operations are still in progress.

Some drivers handle request function re-entrance by maintaining an internal request queue. The request function simply removes any new requests from the I/O request queue and adds them to the internal queue, which is then processed through a combination of task schedulers and interrupt handlers.

One other detail regarding the behavior of the I/O request queue is relevant for block drivers that deal with clustering. It has to do with the queue head: the first request on the queue. For historical compatibility reasons, the operating system (almost) always assumes that a block driver is processing the first entry in the request queue, hence to avoid corruption resulting from conflicting activity, the operating system will never modify a request once it gets to the head of the queue. No further clustering will happen on that request, and the elevator code will not put other requests in front of it.

The queue is designed with physical disk drives in mind. With disks, the amount of time required to transfer a block of data is typically quite small. The amount of time required to position the head (seek) to do that transfer, however, can be very large. Thus, the operating system works to minimize the number and extent of the seeks performed by the device.

Two things are done to achieve these goals. One is the clustering of requests to adjacent sectors on the disk. Most modern file systems will attempt to lay out files in consecutive sectors; as a result, requests to adjoining parts of the disk are common. The operating system also applies an "elevator" algorithm to the requests. An elevator in a skyscraper goes either up or down; it will continue to move in one direction until all of its "requests" (people wanting on or off) have been satisfied. In the same way, the operating system tries to keep the disk head moving in the same direction for as long as possible; this approach tends to minimize seek times while ensuring that all requests get satisfied eventually. Requests are not made of random lists of buffers; instead, all of the buffer heads attached to a single request will belong to a series of adjacent blocks on the disk. Thus a request is, in a sense, a single operation referring to a (perhaps long) group of blocks on the disk. This grouping of blocks is called clustering.

16.3.4 I/O operations

(1) Interrupt-driven I/O

In an interrupt-driven I/O, the dedicated I/O microprocessors can conduct I/O operations. Whenever a data transfer to or from the managed hardware might be delayed for any reason, the driver writer should implement buffering. Data buffers help to detach data transmission and reception from the write and read system calls, and overall system performance benefits.

A good buffering mechanism leads to interrupt-driven I/O, in which an input buffer is filled at interrupt time and is emptied by processes that read the device; an output buffer is filled by processes that write to the device, and is emptied at interrupt time. For interrupt-driven data transfer to happen successfully, the hardware should be able to generate interrupts with the following semantics:

(a) For input, the device interrupts the microprocessor when new data have arrived and are ready to be retrieved. The actual actions to perform depend on whether the device uses I/O ports, memory mapping, or DMA (direct memory access).

(b) For output, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they have finished with the buffer.

However, interrupt-driven I/O introduces the problem of synchronizing concurrent access to shared data items and all the issues related to race conditions. This related topic was discussed in the previous subsection.

(2) Memory-mapped read and write

Memory-mapped and port-mapped I/O are two complementary methods of performing input and output between the CPU and I/O devices.

Memory-mapped I/O uses the same bus to address both memory and I/O devices. CPU instructions used to read and write to memory are also used in accessing I/O devices. To accommodate the I/O devices, areas of CPU addressable space must be reserved for I/O rather than memory, which needs CPU hardware support. This does not have to be permanent; for example, the Commodore 64 could bank switch between its I/O devices and regular memory. The I/O devices monitor the CPU's address bus and respond to any CPU access of their assigned address space, mapping the address to their hardware registers.

Port-mapped I/O uses a special class of CPU instructions. This is generally found on Intel microprocessors, specifically the IN and OUT instructions, which can read and write a single byte to an I/O device. I/O devices have a separate address space from general memory, either from an extra I/O pin on the CPU's physical interface, or an entire bus dedicated to I/O.

(3) Bus-based read and write

In bus-based I/O, the microprocessor has a set of address, data, and control ports corresponding to bus lines, and uses the bus to access memory as well as peripherals. The microprocessor has the bus protocol built into its hardware. Alternatively, the bus may be equipped with memory read and write plus input and output command lines. Now, the command line specifies whether the address refers to a memory location or an I/O device. The full range of addresses may be available for both. Again, with 16 address lines, the system may now support both 64K memory locations and 64K I/O addresses. Because the address space for I/O is isolated from that for memory, this is referred to as isolated I/O, also known as mapped I/O or standard I/O.

16.4 INTERRUPTS

16.4.1 Interrupt overview

An interrupt is a hardware signal from a device to the associated CPU, telling it that the device needs attention. When a device asserts its interrupt request signal, it must be processed in an orderly fashion.

All CPUs, and other intelligent electronic devices, have some mechanism for enabling and disabling interrupt recognition and processing. At the device level, there is usually an interrupt control register with bits to enable or disable the interrupts that the device can generate. At the CPU level, a global mechanism functions to inhibit and enable (often called the global interrupt enable)

recognition of interrupts. If the CPU is available, (it is not doing something else, such as servicing a higher-priority interrupt) it suspends the currently running task or thread, and then invokes the interrupt handler specified for that device within a required time gap. The job of the interrupt handler is to serve for the device, and stop it from interrupting. Once the handler returns, the CPU resumes whatever it was doing before the interrupt occurred.

Systems with multiple interrupt inputs provide the ability to mask (inhibit) interrupt requests individually and/or on a priority basis. This capability may be built into the CPU or provided by an external interrupt controller. Typically, there are one or more interrupt mask registers, with individual bits allowing or inhibiting individual interrupt sources. There is often also one non-maskable interrupt input to the CPU that is used to signal important conditions such as pending power fail, reset button pressed, or watchdog timer expiration.

(1) Interrupt specifications

An interrupt specification, also termed interrupt routing, is the information the system needs in order to link an interrupt handler with a given interrupt, and describes the information provided by the hardware to the system when making an interrupt request. Interrupt specifications typically includes a bus-interrupt level. For vectored interrupts it includes an interrupt vector.

When registering interrupts, the driver must provide the system with an interrupt number. This identifies which interrupt specification the driver is registering a handler for. Most devices have one interrupt; interrupt number zero, but there are devices that have different interrupts for different events. A communications controller may have one interrupt for receive ready and one for transmit ready. The device driver normally knows how many interrupts the device has, but if the driver has to support several variations of a controller it can call to find out the number of device interrupts. For a device with n interrupts, the interrupt numbers range from 0 to n−1.

Buses prioritize device interrupts at one of several bus-interrupt levels. These are then associated with different processor-interrupt levels. A bus-interrupt level that maps to a CPU interrupt priority level above the scheduler priority level is called a high-level interrupt. They must be handled without using system services that manipulate tasks or threads. A bus-interrupt level by itself does not determine whether a device interrupts at high level: a given bus-interrupt level may map to a high-level interrupt on one platform, but map to an ordinary interrupt on another. The driver can choose whether or not to support high-level interrupts, but it always has to check because it cannot assume that its interrupts are not high-level.

(2) Types of interrupts

There are two common ways to request an interrupt: vectored and polled. Both methods commonly specify a bus-interrupt priority level. Their only difference is that vectored devices also specify an interrupt vector, but polled devices do not.

(a) Vectored interrupts

Devices that use vectored interrupts are assigned an interrupt vector. This is a number that identifies that particular interrupt handler. This vector may be fixed, configurable (using jumpers or switches), or programmable. In the case of programmable devices, an interrupt device cookie is used to program the device interrupt vector. When the interrupt handler is registered, the kernel saves the vector in a table.

When the device interrupts, the system enters the interrupt acknowledge cycle, asking the interrupting device to identify itself. The device responds with its interrupt vector, and the kernel then uses this vector to find the responsible interrupt handler.

(b) Polled interrupts

In polled (or auto-vectored) devices, the only information the system has about a device interrupt is its bus-interrupt priority level. When a handler is registered, the system adds it to the list of potential interrupt handlers for the bus-interrupt level. When an interrupt occurs, the system must determine which device, of all the devices at that level, actually interrupted. It does this by calling all the interrupt handlers for that bus-interrupt level until one of them claims the interrupt.

(c) Software interrupts

Also known as soft interrupts, they are initiated by software rather than a device. Handlers for these interrupts must also be added to and removed from the system. Soft interrupt handlers run in interrupt context and therefore can be used to do many of the tasks that belong to an interrupt handler.

(3) Interrupt handlers

Modern CPUs use a general mechanism for processing exceptions, traps, and interrupts: an interrupt vector table. Some contain only the address of the code to be executed. In most cases, a specific ISR (interrupt service routine) is responsible for servicing each interrupting device and acknowledging, clearing, and rearming its interrupt; sometimes servicing the device (for example, reading data from a serial port) automatically clears and rearms the interrupt.

Hardware interrupt handlers are very quick, since they may suspend other system activity while running (particularly in high-level interrupt handlers). For example, they may prevent lower-priority interrupts from occurring while they run. For this reason they should do the minimum amount of work needed to service the device.

Software interrupt handlers run at a lower priority than hardware versions, so they can do more work without seriously impacting the performance of the system. If the hardware interrupt handler is high level, it is severely restricted in what it can do. In this case, it is a good idea to simply trigger a software interrupt in the high-level handler and put all possible processing into this handler. Software interrupt handlers must not assume that they have work to do when they run, since (like hardware interrupt handlers) they can run because some other driver triggered a soft interrupt. For this reason, the driver must indicate to the soft interrupt handler that it should do work before triggering the soft interrupt.

(4) Interrupt latency

Interrupts may occur at any time, but the CPU does not instantly recognize and process them. First, the CPU will not recognize a new interrupt while interrupts are disabled. Second, the CPU must, upon recognition, stop fetching new instructions and complete those still in progress.

Because the interrupt is totally unrelated to the program it interrupts, the CPU and ISR work together to save and restore the full state of the interrupted program (stack, flags, registers, and so on). The running program is not affected by the interruption, although it takes longer to execute.

Many interrupt controllers provide a means of prioritizing interrupt sources, so that, in the event of multiple interrupts occurring at (approximately) the same time, the more time-critical ones are processed first. These same systems usually also provide for prioritized interrupt handling, a means by which a higher-priority interrupt can interrupt the processing of a lower-priority interrupt. This is called interrupt nesting. In general, the ISR should only take care of the time-critical portion of the processing, then, depending on the complexity of the system, it may set a flag for the main loop, or use an operating system call to awaken a task to perform the non-time-critical portion.

The longest-period global interrupt recognition is inhibited. The longest-period global interrupt recognition may be due to these time factors: for example,the time it would take to execute all higher-priority interrupts if they occurred simultaneously; the time it takes the specific ISR to service all of its interrupt requests (if multiple ones are possible); the time it takes to finish the program instructions in progress and save the current program state and begin the ISR.

16.4.2 Interrupt handling

Handling interrupts is at the heart of a real-time and embedded control system. The actual process of determining a good handling method can be complicated, since numerous actions are occurring simultaneously at a single point, and have to be handled rapidly and efficiently. This subsection will provide a practical guide to designing an interrupt handler, and will discuss the various trade-offs between the different methods. The methods covered are (1) non-nested interrupt handler, (2) nested interrupt handler, (3) re-entrant nested interrupt handler, and (4) prioritized interrupt handlers.

(1) Non-nested interrupt handler

The simplest interrupt handler is a handler that is non-nested. This means that while handling an interrupt, all others are disabled until control is returned to the interrupted task or process. This type can only service a single interrupt at a time, and so are not suitable for complex embedded systems that service multiple interrupts with differing priority levels.

When the "Interrupt request" pin is raised, the microprocessor will disable further interrupts. It will then set the controller to point to the correct entry in the vector table and execute that instruction, which will alter the controller to point to the interrupt handler.

Once in the interrupt code, the interrupt handler has to first save the context, so that it can be restored on return. The handler can then identify the interrupt source and call the appropriate ISR. After servicing the interrupt the context can be restored, and the controller manipulated to point back to the next instruction before the interruption. Figure 16.8 shows the various stages that occur when an interrupt is raised in a system that has implemented a simple non-nested interrupt handler. Each stage is explained in more detail below:

(a) External source (e.g., from an interrupt controller) sets the interrupt flag. Processor masks further external interrupts and vectors to the interrupt handler through an entry in the vector table.
(b) On entry to the handler, the handler code saves the current context of the non-banked registers.
(c) The handler then identifies the interrupt source and executes the appropriate ISR.
(d) ISR services the interrupt.
(e) On return from the ISR the handler restores the context.
(f) Enables interrupts and return.

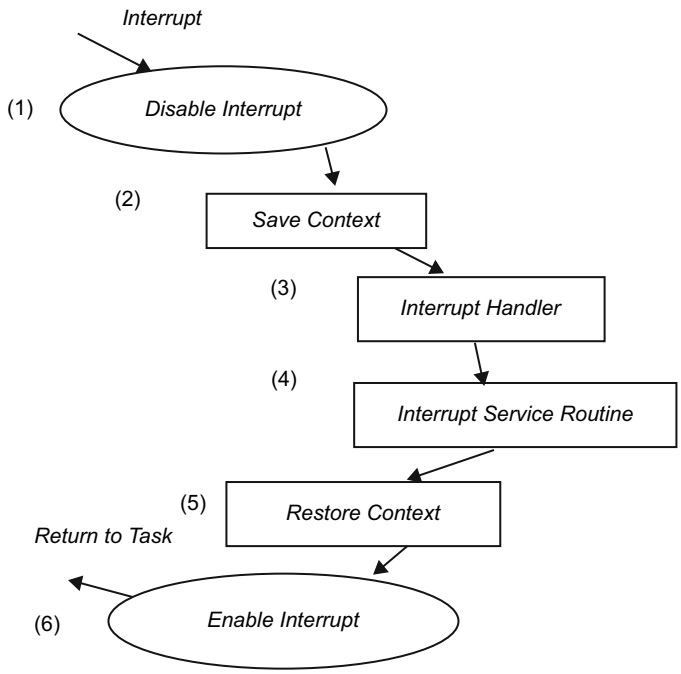


FIGURE 16.8

A simple non-nested interrupt handler.

(2) Nested interrupt handler

A nested interrupt handler allows another interrupt to occur within the currently called handler. This is achieved by re-enabling interrupts before the handler has fully serviced the current one. This feature increases the complexity of the system, and so introduces the possibility of subtle timing issues that can cause system failure. These subtle problems can be extremely difficult to resolve, so the nested interrupt method has to be designed carefully. This is achieved by protecting the context restoration from interruption, so that the next interrupt will not fill (overflow) the stack, or corrupt any of the registers.

Standard problems can occur if nested interrupts are supported. One of these is a race condition, where a cascade of interrupts occurs, which will cause a continuous interruption of the handler until either the interrupt stack is full (overflowing) or the registers are corrupted. A designer has to balance efficiency with safety. This involves using a defensive coding style that assumes problems will occur, and check the stacking and protecting against register corruption where possible.

Figure 16.9 shows a nested interrupt handler. As can been seen from the diagram, the handler is quite a bit more complicated than the simple non-nested interrupt handler described above.

How stacks are organized is one of the first decisions a designer has to make when designing a nested interrupt handler. There are two fundamental methods that can be adopted either using

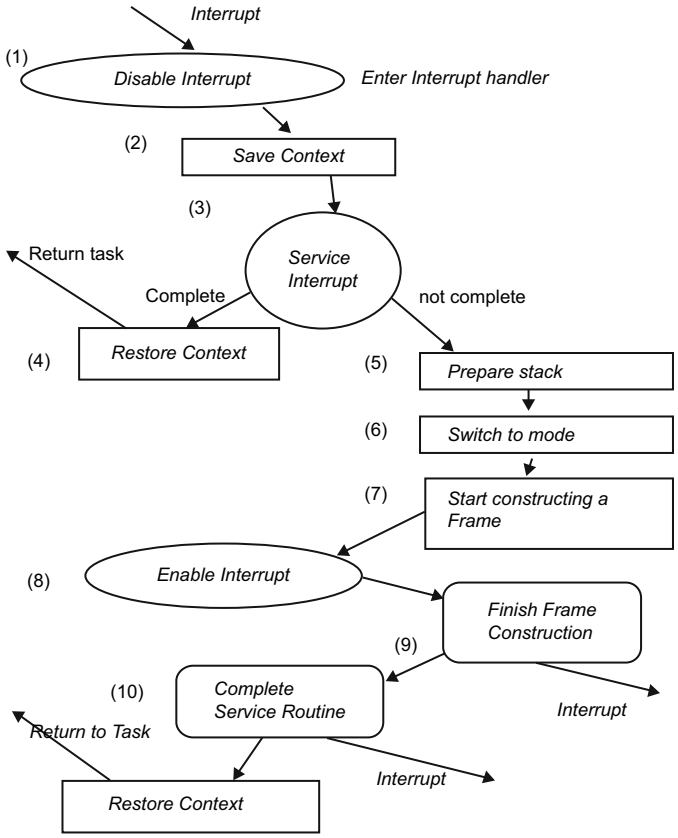


FIGURE 16.9

A complex nested interrupt handler.

a single stack or multiple stacks. The multiple-stack method uses one stack for each interrupt or service routine. This increases the execution time and complexity of the handler. For a time-critical system, these tend to be undesirable characteristics.

The nested interrupt handler entry code is identical to the simple non-nested interrupt handler, except that on exit the handler tests a flag that is updated by the ISR that indicates whether further processing is required. If not, then the service routine is complete and the handler can exit. Otherwise the handler can take several actions; re-enabling interrupts and/or performing a context switch. Re-enabling interrupts involves switching out of interrupt request (IRQ) mode. We cannot simply re-enable interrupts in IRQ mode as this would lead to the link register being corrupted if an interrupt occurred after a branch with link instruction. This problem will be discussed in more detail below.

As a side note, performing a context switch involves flattening (emptying) the IRQ stack, as the handler should not perform a context switch while there are data on it unless the handler can maintain

a separate IRQ stack for each task, which is, as mentioned previously, undesirable. All registers saved on the IRQ stack must be transferred to the task's stack. The remaining registers must then be saved on the task stack. This is transferred to a reserved block on the stack called a stack frame.

(3) Re-entrant nested interrupt handler

A re-entrant nested interrupt handler is a method of handling multiple interrupts that are filtered by priority (Figure 16.10). This is important since there is a requirement that interrupts with higher priority have a lower latency. This type of filtering cannot be achieved using the conventional nested interrupt handler. The basic difference between this type and a nested interrupt handler is that the interrupts are re-enabled early on in the interrupt handler to achieve low interrupt latency. There are a number of issues relating to re-enabling the interrupts early, which are described in more detail in the following paragraphs.

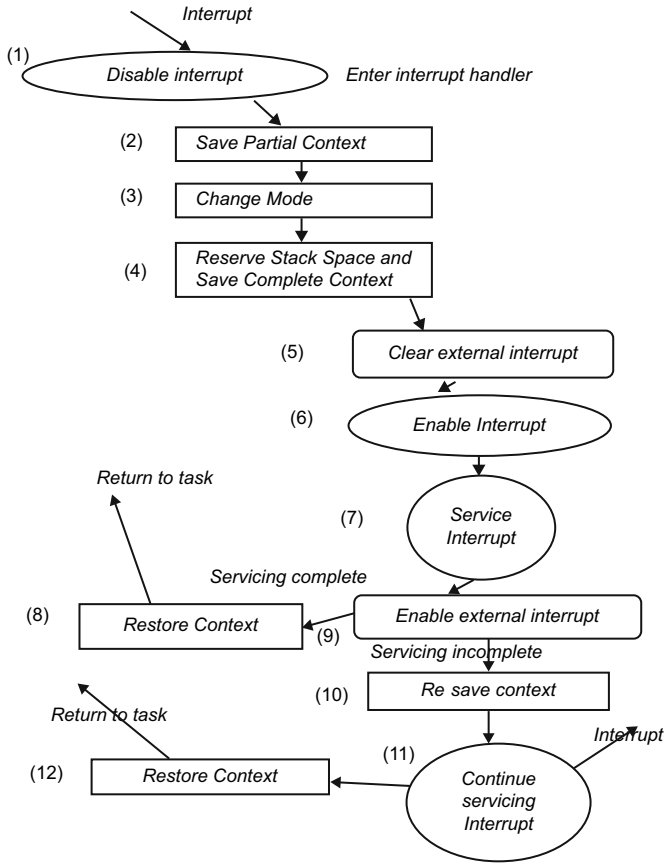


FIGURE 16.10

A re-entrant nested interrupt handler.

If interrupts are re-enabled in an interrupt mode, and the interrupt routine performs a subroutine call instruction (BL), the subroutine return address will be set in a special register. This address would be subsequently destroyed by an interrupt, which would overwrite the return address into this special register. To avoid this, the interrupt routine should swap into SYSTEM mode. The BL instruction can then use another register to store the subroutine address. The interrupts must be disabled at the source by setting a bit in the interrupt controller before re-enabling interrupts through the current processor status register (CPSR). If interrupts are re-enabled in the CPSR before processing is complete and the interrupt source is not disabled, an interrupt will be immediately regenerated, leading to an infinite interrupt sequence or race condition. Most interrupt controllers have an interrupt mask register that allows you to mask out one or more interrupts, leaving the remainder of the interrupts enabled.

The interrupt stack is unused since interrupts are serviced in SYSTEM mode (i.e., on the task's stack). Instead the IRQ stack pointer is used to point to a 12-byte structure used to store some registers temporarily on interrupt entry. It is paramount for a re-entrant nested interrupt handler to operate effectively so the interrupts can be prioritized, otherwise the system latency degrades to that of a nested interrupt handler, because lower-priority interrupts will be able to preempt the servicing of a higher-priority interrupt. This can lead to the locking out of higher-priority interrupts for the duration of the servicing of a lower-priority interrupt.

(4) Prioritized interrupt handler

There are four types of prioritized interrupt handler which provide different handling strategies, as given below:

(a) Simple prioritized interrupt handler

Both the simple and nested interrupt handler service interrupts on a first-come first-served basis, whereas a prioritized interrupt handler will associate a priority level with a particular interrupt source. This is used to dictate the order that the interrupts will be serviced, and means that a higher-priority interrupt will take precedence over a lower-priority interrupt, which is desirable in an embedded system.

Prioritization can be achieved either in hardware or in software. Hardware prioritization means that the handler is easier to design, since the interrupt controller will provide the current highest-priority interrupt which requires servicing. These systems require more initialization code, since the interrupts and associated priority-level tables have to be constructed before the system can be switched on. Software prioritization requires an external interrupt controller, which has to provide a minimal set of functions, including being able to set and unset masks and read interrupt status and source. For software systems the rest of this subsection will describe a priority interrupt handler, and to help with this a fictional interrupt controller will be used. With the Intel Pentium series, the interrupt controller takes in multiple interrupt sources and will generate an IRQ and/or FIQ signal depending on whether a particular interrupt source is enabled or disabled.

The interrupt controller has a register that holds the raw interrupt status (IRQRawStatus). A raw interrupt is an interrupt that has not been masked by a controller. IRQEnable register determines which interrupts are masked from the microprocessor. This register can only be set or cleared using IRQEnableSet and IRQEnableClear. Table 16.2 shows a summary of the register set names and the types of operation (read/write) that can occur with it. Most interrupt controllers also have

Table 16.2 Interrupt Controller Registers

Register	R/W	Description
IRQRawStatus	r	Represents interrupt sources that are actively HIGH
IRQEnable	r	Masks the interrupt sources that generates IRQ/FIQ to the CPU
IRQStatus	r	Represents interrupt sources after masking
IRQEnableSet	w	Sets the interrupt enable register
IRQEnableClear	w	Clears the interrupt enable register

a corresponding set of registers for FIQ; some can also be programmed to select the type of interrupt distinction, i.e., select the type of interrupt raised (IRQ/FIQ) from a particular interrupt source.

(b) Standard prioritized interrupt handler

A simple priority interrupt handler tests all the interrupts to establish the highest priority. An alternative solution is to branch early when the highest-priority interrupt has been identified. The prioritized interrupt handler follows the same entry code as for the simple prioritized interrupt handler. The prioritized interrupt handler has the same start as a simple prioritized handler but intercepts the interrupts with a higher priority earlier. Figure 16.11 gives the part of the algorithm applied to the standard prioritized interrupt handler.

(c) Direct prioritized interrupt handler

A direct prioritized interrupt handler branches directly to the ISR. Each of these is responsible for disabling lower-priority interrupts before modifying the CPSR so that interrupts are re-enabled. This type of handler is relatively simple, since the masking is done by the service routine. This causes minimal duplication of code since each service routine is effectively carrying out the same task.

(d) Grouped prioritized interrupt handler

Lastly, the grouped priority interrupt handler is assigned a group priority level for a set of interrupt sources, sometimes important when there is a large number of them. It tends to reduce the complexity of the handler since it is not necessary to scan through every interrupt to determine the priority level. This may improve the response times.

16.4.3 Enable and disable interrupts

Interrupt masking allows us to disable the detection of an interrupt-line assertion, which causes the operating system or kernel to ignore an interrupt signal. The signal can be ignored at the microprocessor level (or CPU level) or at other levels in the hardware architecture. In some cases, each interrupt source in the system can be masked individually. In other cases, masking an interrupt in a microprocessor register can mask a group of interrupts. An example of this is multiple interrupts.

When an interrupt occurs, the microprocessor must globally disable interrupts at the microprocessor level to avoid being interrupted while gathering and saving interrupt state information. Because disabling interrupts globally blocks all other interrupts, masking should take place for as

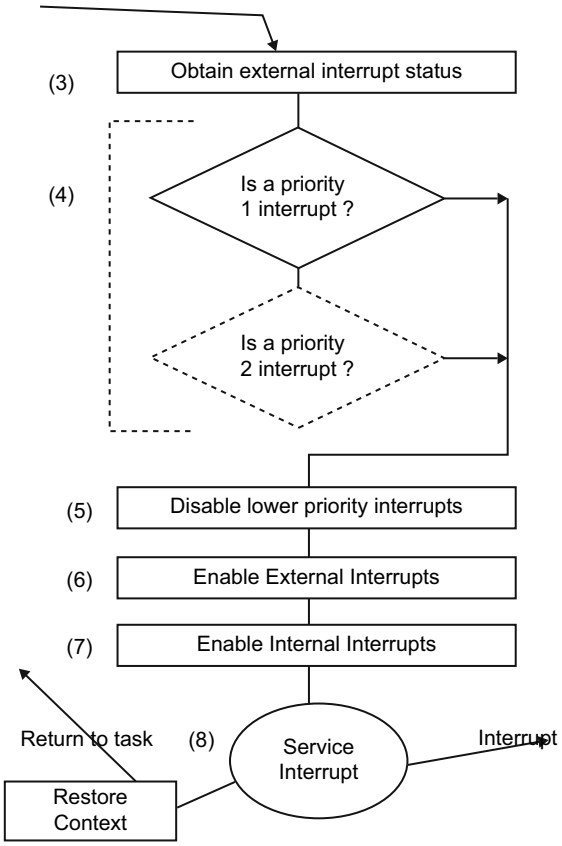


FIGURE 16.11

Part of a prioritized interrupt handler.

short a time as possible. When you determine what has specifically interrupted, you can mask just that interrupt.

A maskable interrupt is essentially a hardware interrupt that may be ignored by setting a bit in an interrupt mask register's (IMR) bit-mask. Similarly, a non-maskable interrupt is a hardware interrupt that typically does not have a bit-mask associated with it.

All the regular interrupts that we normally use and refer to by number are called maskable interrupts. The processor is able to mask, or temporarily ignore, any interrupt if it needs to, to finish something else that it is doing. In addition, however, there is a non-maskable interrupt (NMI) that can be used for serious conditions that demand the microprocessor's immediate attention. The NMI cannot be ignored by the system unless it is shut off specifically.

When an NMI signal is received, the microprocessor immediately drops whatever it was doing and attends to it. As you can imagine, this could cause crashes if used improperly. In fact, the NMI signal is normally used only for critical problem situations, such as serious hardware errors. Its most common

use is to signal a parity error from the memory subsystem, which must be dealt with immediately to prevent possible data corruption.

16.4.4 Interrupt vector

As mentioned in Chapter 5, the interrupt vector table can start at memory address 0x00000000. A vector table consists of a set of assembler (or machine) instructions, which cause the controller or computer to jump to a specific location that can handle a specific exception or interrupt. A vector uses a special assembler instruction to load the address of the handler. The address of the handler will be called indirectly, whereas a branch instruction will go directly to the handler. When booting a system, quite often the read-only memory (ROM) is located at 0x00000000. This means that when SRAM is remapped to location 0x00000000 the vector table has to be copied to SRAM at its default address prior to the remap, normally achieved by the system initialization code. SRAM is normally remapped because it is wider and faster than ROM; it also allows vectors to be dynamically updated as requirements change during program execution.

Figure 16.12 shows a typical vector table of a real system. The undefined instruction handler is located so that a simple branch is adequate, whereas the other vectors require an indirect address using assembler instructions specific for loading.

Where the interrupt stack is placed depends on the real-time operating system (RTOS) requirements and the specific hardware being used. The example in Figure 16.13 shows two possible stack layouts, firstly (A) shows the traditional stack layout with the interrupt stack being stored underneath the code segment. The second, layout (B), shows the interrupt stack at the top of the memory above the user stack. This has the advantage that the stack grows into the user stack and thus does not corrupt the vector table. For each mode, a stack has to be set up, carried out every time the processor is reset.

If the interrupt stack expands into the interrupt vector, the target system will crash, unless some means exists to handle this error when it occurs.

Before an interrupt can be enabled, the IRQ mode stack has to be set up, normally, in the initialization code for the system. It is important that the maximum size of the stack is known, since that size can be reserved for the interrupt stack.

```
0x00000000:  0xe59ffa38  8... : > ldr  pc,0x00000a40
0x00000004:  0xea000502  .... :   b    0x1414
0x00000008:  0xe59ffa38  8... :   ldr  pc,0x00000a48
0x0000000c:  0xe59ffa38  8... :   dr   pc,0x00000a4c
0x00000010:  0xe59ffa38  8... :   ldr  pc,0x00000a50
0x00000014:  0xe59ffa38  8... :   ldr  pc,0x00000a54
0x00000018:  0xe59ffa38  8... :   ldr  pc,0x00000a58
0x0000001c:  0xe59ffa38  8... :   ldr  pc,0x00000a5c
```

FIGURE 16.12

A typical interrupt vector table.

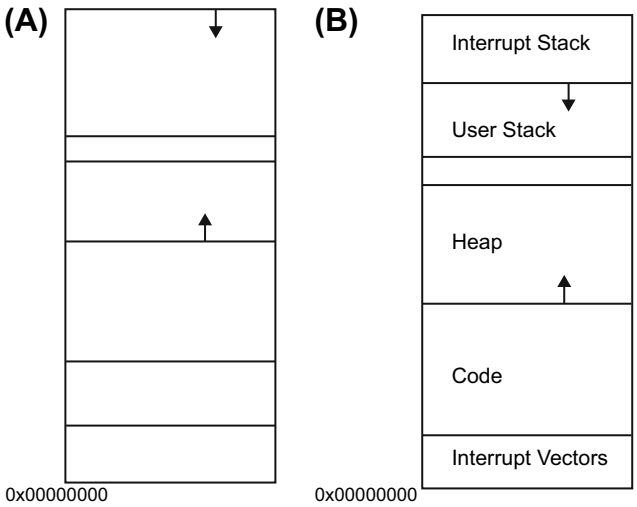


FIGURE 16.13

Typical stack design layouts.

16.4.5 Interrupt service routines

An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt. ISR examines an interrupt and determines how to handle it executes the handling, and then returns a logical interrupt value. If no further handling is required the ISR notifies the kernel with a return value. An ISR must perform very quickly to avoid slowing down the operation of the device and the operation of all lower-priority ISRs.

Although an ISR might move data from a CPU register or a hardware port into a memory buffer, in general it relies on a dedicated interrupt thread (or task), called the interrupt service thread (IST), to do most of the required processing. If additional processing is required, the ISR returns a logical interrupt value to the kernel. It then maps a physical interrupt number to a logical interrupt value. For example, the keyboard might be associated with hardware interrupt 4 on one device and hardware interrupt 15 on another device. The ISR translates the hardware-specific value to the standard value corresponding to the specific device.

When an ISR notifies the kernel of a specific logical interrupt value, the kernel examines an internal table to map the logical interrupt value to an event handle. The kernel then wakes the IST by signaling the event. An event is a standard synchronization object that serves as an alarm clock to wake up a thread when something interesting happens.

The interrupt service thread is a thread that does most of the interrupt processing. The operating system wakes the IST when it has an interrupt to process, otherwise, it remains idle. For the operating system to wake the IST, the IST must associate an event object with an interrupt identifier. After an interrupt is processed, the IST should wait for the next interrupt signal. This call is usually inside a loop.

When a hardware interrupt occurs, the kernel signals the event on behalf of the ISR, and then the IST performs necessary I/O operations in the device to collect the data and process them. When the interrupt processing is completed, the IST informs the kernel to re-enable the hardware interrupt.

An interrupt notification is a signal from an IST that notifies the operating system that an event must be processed. For devices that connect to a platform through intermediate hardware, the device driver for that hardware should pass the interrupt notification to the top-level device driver. Generally, the intermediate hardware's device driver has some facility that allows another device driver to register a call-back function, which the intermediate device driver calls when an interrupt occurs.

As an example, PC cards connect to hardware platforms through a PC card slot, which is an intermediate piece of hardware with its own device driver. When a PC card device sends an interrupt, it is actually the PC card slot hardware that signals the physical interrupt on the system bus. The device driver for the PC card slot has an ISR and IST that handle the physical interrupt. They use a function to pass the interrupt to the device driver for the PC card device. Devices with similar connection methods behave similarly.

16.5 MEMORY MANAGEMENT

Memory management is one of the most important subsystems of any operating system for computer control systems, and is even more critical in a RTOS than in standard operating systems. Firstly, the speed of memory allocation is important in a RTOS. A standard memory allocation scheme scans a linked list of indeterminate length to find a free memory block; however, memory allocation has to occur in a fixed time in a RTOS. Secondly, memory can become fragmented as free regions become separated by regions that are in use, causing a program to stall, unable to get memory, even though there is theoretically enough available. Memory allocation algorithms that slowly accumulate fragmentation may work perfectly well for desktop machines rebooted every day or so but are unacceptable for embedded systems that often run for months without rebooting.

Memory management is the process by which a computer control system allocates a limited amount of physical memory among its various processes (or tasks) in a way that optimizes performance. Actually, each process has its own private address space, initially divided into three logical segments: text, data, and stack. The text segment is read-only and contains the machine instructions of a program, the data and stack segments are both readable and writable. The data segment contains the initialized and non-initialized data portions of a program, whereas the stack segment holds the application's run-time stack. On most machines, this is extended automatically by the kernel as the process executes. This is done by making a system call, but change to the size of a text segment only happens when its contents are overlaid with data from the file system, or when debugging takes place. The initial contents of the segments of a child process are duplicates of the segments of its parent.

The contents of a process address space do not need to be completely in place for a process to execute. If a process references a part of its address space that is not resident in main memory, the system pages the necessary information into memory. When system resources are scarce, the system

uses a two-level approach to maintain available resources. If a modest amount of memory is available, the system will take memory resources away from processes if these resources have not been used recently. Should there be a severe resource shortage, the system will resort to swapping the entire context of a process to secondary storage. This paging and swapping done by the system are effectively transparent to processes, but a process may advise the system about expected future memory utilization as a performance aid.

A common technique for doing the above is virtual memory, which simulates a much larger address space than is actually available, using a reserved disk area for objects that are not in physical memory. The operating system's kernel often performs memory allocations that are needed for only the duration of a single system call. In a user process, such short-term memory would be allocated on the run-time stack. Because the kernel has a limited run-time stack, it is not feasible to allocate even moderately-sized blocks of memory on it, so a more dynamic mechanism is needed. For example, when the system must translate a path name, it must allocate a 1-kbyte buffer to hold the name. Other blocks of memory must be more persistent than a single system call, and thus could not be allocated on the stack even if there was space. An example is protocol-control blocks that remain throughout the duration of a network connection.

This section discusses virtual memory techniques, memory allocation and deallocation, memory protection and memory access control.

16.5.1 Virtual memory

When it is executing a program, the microprocessor reads an instruction from memory and decodes it. At this point it may need to fetch or store the contents of a location in memory, so it executes the instruction and then moves on to the next. In this way the microprocessor is always accessing memory, either to fetch instructions or to fetch and store data. In a virtual memory system all of these addresses are virtual, and not physical addresses. They are converted into physical addresses by the microprocessor, based on information held in a set of tables maintained by the operating system.

The operating system uses virtual memory to manage the memory requirements of its processes by combining physical memory with secondary memory (swap space) on a disk, usually located on a hardware disk drive. Diskless systems use a page server to maintain their swap areas on the local disk (extended memory). The translation from virtual to physical addresses is implemented by a memory management unit (MMU), which may be either a module of the CPU, or an auxiliary, closely coupled chip. The operating system is responsible for deciding which parts of the program's simulated main memory are kept in physical memory, and also maintains the translation tables that map between virtual and physical addresses. This is shown in Figure 16.14.

We will now discuss three techniques of implementing virtual memory; paging, swapping and segmentation.

(1) Paging

Almost all implementations of virtual memory divide the virtual address space of an application program into pages; a page is a block of contiguous virtual memory addresses. Here, the low-order bits of the binary representation of the virtual address are preserved, and used directly as the low-order bits of the actual physical address; the high-order bits are treated as a key to one or more address translation

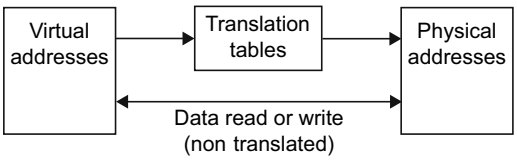


FIGURE 16.14

The address translations between physical and virtual memory.

tables, which provide the high-order bits with the actual physical address. For this reason, a range of consecutive addresses in the virtual address space, whose size is a power of two, will be translated to a corresponding range of consecutive physical addresses. The memory referenced by such a range is called a page. The page size is typically in the range 512 8192 bytes (with 4 kB currently being very common), though 4 MB or even larger may be used for special purposes. (Using the same or a related mechanism, contiguous regions of virtual memory larger than a page are often mappable to contiguous physical memory for purposes other than virtualization, such as setting access and caching control bits.)

Almost all implementations use page tables to translate the virtual addresses seen by the application into physical addresses (also referred to as real addresses) used by the hardware. The operating system stores the address translation tables, i.e. the mappings from virtual to physical page numbers, in a data structure known as a page table. When the CPU tries to reference a memory location that is marked as unavailable, the MMU responds by raising an exception (commonly called a page fault) with the CPU, which then jumps to a routine in the operating system. If the page is in the swap area, this routine invokes an operation called a page swap, to bring in the required page.

The operating systems can have one page table or a separate page table for each application. If there is only one, different applications running at the same time will share a single virtual address space, i.e. they use different parts of a single range of virtual addresses. The operating systems which use multiple page tables provide multiple virtual address spaces, so concurrent applications seem to use the same range of virtual addresses, but their separate page tables redirect to different real addresses.

Figure 16.15 shows the virtual address spaces of two processes, X and Y, each with their own page tables, which map each process's virtual pages into physical pages in memory. This shows that process X's virtual page frame number 0 is mapped into memory at physical page frame number 1 and that process Y's virtual page frame number 1 is mapped into physical page frame number 4. Each entry in the theoretical page table contains the following information: (1) a valid flag that indicates whether this page table entry is valid; (2) the physical page frame number that this entry describes; (3) the access control information that describes how the page may be used.

(2) Swapping

Swap space is a portion of hard disk used for virtual memory that is usually a dedicated partition (i.e., a logically independent section of a hard disk drive), created during the installation of the operating system. Such a partition is also referred to as a swap partition. However, swap space can also be

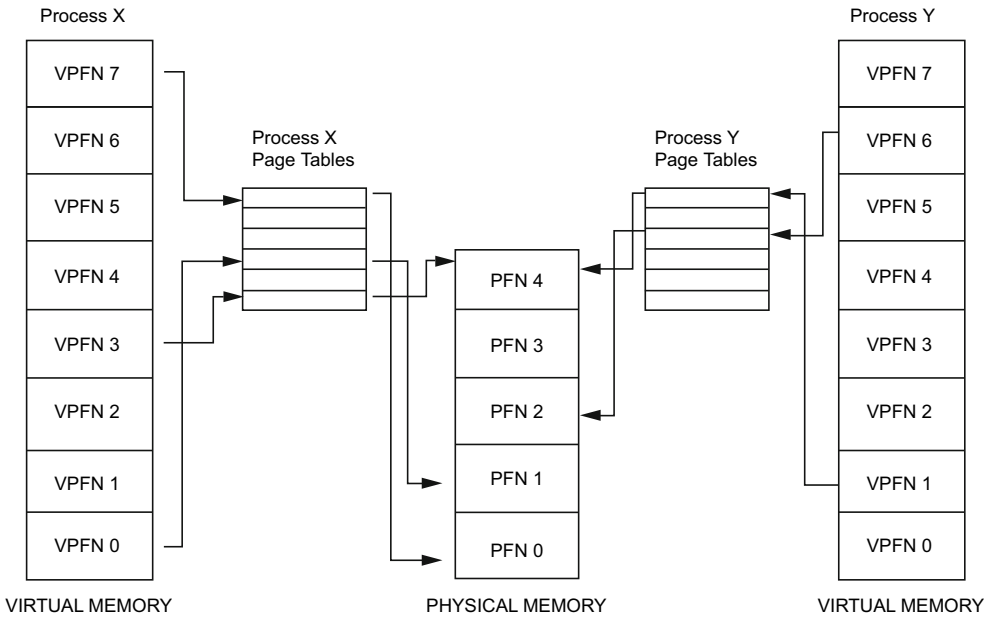


FIGURE 16.15

An abstract model for mapping virtual addresses to physical addresses in the implementation of virtual memory.

a special file. Although it is generally preferable to use a swap partition rather than a file, sometimes it is not practical to add or expand a partition when the amount of RAM is being increased. In such case, a new swap file can be created with a system call to mark a swap space.

It is also possible for a virtual page to be marked as unavailable because the page was never previously allocated. In such cases, a page of physical memory is allocated and filled with zeros, the page table is modified to describe it, and the program is restarted as above. Figure 16.16 illustrates how the virtual memory of a process might correspond to what exists in physical memory, on swap, and in the file system. The U-area of a process consists of two 4 kB pages (displayed here as U1 and U1) of virtual memory containing information about the process that is needed by the system during execution. In this example, these pages are shown in physical memory, and the data pages, D3 and D4, are shown as being paged out to the swap area on disk. The text page, T4, has also been paged out, but it is not written to the swap area as it exists in the file system. Those pages that have not yet been accessed by the process (D5, T2, and T5) do not occupy any resources in physical memory or in the swap area.

The page swap operation involves a series of steps. Firstly it selects a page in memory; for example, a page that has not been recently accessed and (preferably) has not been modified since it was last read. If the page has been modified, the process writes the modified page to the swap area. The next step in the process is to read in the information in the needed page (the page corresponding to the virtual address the original program was trying to reference when the exception occurred) from the swap file. When the page has been read in, the tables for translating virtual

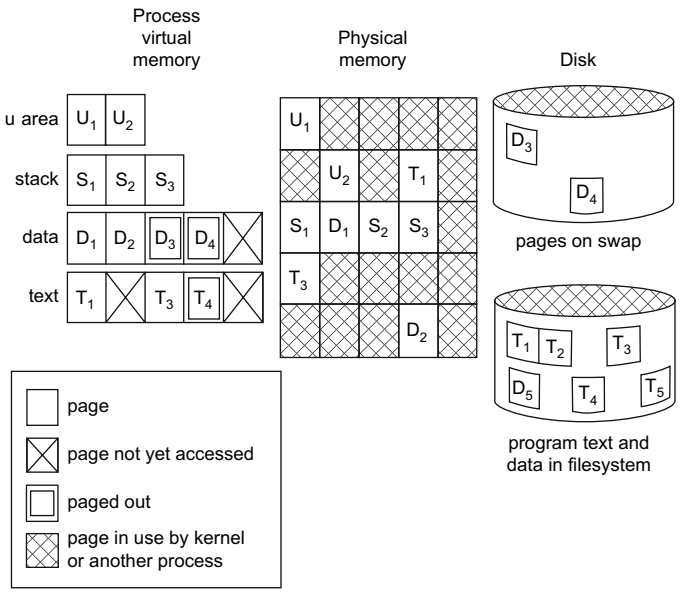


FIGURE 16.16

An illustration of page swapping for implementing virtual memory.

addresses to physical addresses are updated to reflect the revised contents of physical memory. Once the page swap completes, it exits, the program is restarted and returns to the point that caused the exception.

(3) Segmentation

Some operating systems do not use paging to implement virtual memory, but use segmentation instead. For an application process, segmentation divides its virtual address space into variable-length segments, so a virtual address consists of a segment number and an offset within the segment.

Memory is always physically addressed with a single number (called absolute or linear address). To obtain it, the microprocessor looks up the segment number in a table to find a segment descriptor. This contains a flag indicating whether the segment is present in main memory and, if so, the address of its starting point (segment's base address) and its length. It checks whether the offset within the segment is less than the length of the segment and, if not, generates an interrupt. If a segment is not present in main memory, a hardware interrupt is raised to the operating system, which may try to read the segment into main memory, or to swap it in. The operating system may need to remove other segments (swap out) in order to make space for the segment to be read in.

The difference between virtual memory implementations that use pages and those using segments is not only about the memory division. Sometimes the segmentation is actually visible to the user processes, as part of the semantics of the memory model. In other words, instead of a process just

having a memory which looks like a single large vector of bytes or words, it is more structured. This is different from using pages, which does not change the model visible to the process. This has important consequences.

It is possible to combine segmentation and paging, usually by dividing each segment into pages. In such systems, virtual memory is usually implemented by paging, with segmentation used to provide memory protection. The segments reside in a 32-bit linear paged address space, which segments can be moved into and out of, and pages in that linear address space can be moved in and out of main memory, providing two levels of virtual memory. This is quite rare, however, most systems only use paging.

16.5.2 Memory allocation and deallocation

The inefficient allocation or deallocation of memory can be detrimental to system performance. The presence of wasted memory in a block is called internal fragmentation, and it occurs because the size that was requested was smaller than that allocated. The result is a block of unusable memory, which is considered as allocated when not being used. The reverse situation is called external fragmentation, when blocks of memory are freed, leaving non-contiguous holes. If these holes are not large, they may not be usable because further requests for memory may call for larger blocks. Both internal and external fragmentation results in unusable memory.

Memory allocation and deallocation is a process that has several layers of application. If one application fails, another operates to attempt to resolve the request. This whole process is called dynamic memory management in C++ or C. Memory allocation is controlled by a subsystem called malloc, which controls the heap, a region of memory to which memory allocation and deallocation occurs. The reallocation of memory is also under the control of malloc. In malloc, the allocation of memory is performed by two subroutines, malloc and calloc. Deallocation is performed by the free subroutine, and reallocation is performed by the subroutine known as realloc. In deallocation, those memory blocks that have been deallocated are returned to the binary tree at its base. Thus, a binary tree can be envisioned as a sort of river of information, with deallocated memory flowing in at the base and allocated memory flowing out from the tips.

Garbage collection is another term associated with deallocation of memory. This refers to an automated process that determines what memory is no longer in use, and so recycles it. The automation of garbage collection relieves the user of time-consuming and error-prone tasks. There are a number of algorithms for the garbage collection process, all of which operate independently of malloc.

Memory allocation and deallocation can be categorized as static or dynamic.

(1) Static memory allocation

Static memory allocation refers to the process of allocating memory at compile-time, before execution.

One way to use this technique involves a program module (e.g., function or subroutine) declaring static data locally, such that these data are inaccessible to other modules unless references to them are passed as parameters or returned. A single copy of this static data is retained and is

accessible through many calls to the function in which it is declared. Static memory allocation therefore has the advantage of modularizing data within a program so that it can be used throughout run-time. The use of static variables within a class in object-oriented programming creates a single copy of such data to be shared among all the objects of that class.

(2) Dynamic memory allocation

Dynamic memory allocation is the allocation of memory storage for use during the run-time of a program, and is a way of distributing ownership of limited memory resources among many pieces of data and code. A dynamically allocated object remains allocated until it is deallocated explicitly, either by the programmer or by a garbage collector; this is notably different from automatic and static memory allocation. It is said that such an object has dynamic lifetime.

Memory pools allow dynamic memory allocation comparable to malloc, or the operator "new" in C++. As those implementations suffer from fragmentation because of variable block sizes, it can be impossible to use them in a real-time system due to performance problems. A more efficient solution is to pre-allocate a number of memory blocks of the same size, called the memory pool. The application can allocate, access, and free blocks represented by handles at run-time.

Fulfilling an allocation request, which involves finding a block of unused memory of a certain size in the heap, is a difficult problem. A wide variety of solutions have been proposed, and some of the most commonly used are discussed here.

(a) Free lists

A free list is a data structure used in a scheme for dynamic memory allocation that operates by connecting unallocated regions of memory together in a linked list, using the first word of each unallocated region as a pointer to the next. It is most suitable for allocating from a memory pool, where all objects have the same size.

Free lists make the allocation and deallocation operations very simple. To free a region, it is just added it to the free list. To allocate a region, we simply remove a single region from the end of the free list and use it. If the regions are variable-sized, we may have to search for a large enough region, which can be expensive.

Free lists have the disadvantage, inherited from linked lists, of poor locality of reference and thus poor data cache utilization, and they provide no way of consolidating adjacent regions to fulfill allocation requests for large regions. Nevertheless, they are still useful in a variety of simple applications where a full-blown memory allocator is unnecessary, or requires too much overhead.

(b) Paging

As mentioned earlier, the memory access part of paging is done at the hardware level through page tables, and is handled by the MMU. Physical memory is divided into small blocks called pages (typically 4 kB or less in size), and each block is assigned a page number. The operating system may keep a list of free pages in its memory, or may choose to probe the memory each time a request is made (though most modern operating systems do the former). In either case, when a program makes a request for memory, the operating system allocates a number of pages to it, and keeps a list of allocated pages for that particular program in memory.

16.5.3 Memory protection

The topic of memory management in this section addresses a different set of constructs related to physical and virtual memory: protected memory, infinite amount of memory, and transparent sharing. Table 16.3 shows these illusions for physical memory and virtual memory.

Perhaps the simplest model for using memory is to provide single programming without memory protection, where each process (or task) runs with a range of physical memory addresses. Given that a single-programming environment allows only one process to run at a time, this can use the same physical addresses every time, even across reboots. Typically, processes use the lower memory addresses (low memory), and an operating system uses the higher memory addresses (high memory). An application process can address any physical memory location.

One step beyond the single-programming model is to provide multiprogramming without memory protection. When a program is copied into memory, a linker-loader alters the code of the program (loads, stores, jumps) to use the address of where the program lands in memory. In this environment, bugs in any program can cause other programs to crash, even the operating system.

The third model is to have a multitasking operating system with memory protection, which keeps user programs from crashing one another and the operating system. Typically, this is achieved by two hardware-supported mechanisms: address translation and dual-mode operation.

(1) Address translation

Each process is associated with an address space, or all the physical addresses it can touch. However, each process appears to own the entire memory, with the starting virtual address of 0. The missing piece is a translation table that translates every memory reference from virtual addresses to physical addresses.

Translation provides protection because there is no way for a process to talk about other processes' address, and it has no way of touching the code or data of the operating system. The operating system uses physical addresses directly, and involves no translation. When an exception occurs, the operating system is responsible for allocating an area of physical memory to hold the missing information (and possibly in the process pushing something else out to disk), bringing the relevant information in from the disk, updating the translation tables, and finally resuming execution of the software that incurred the exception.

(2) Dual-mode operation

Translation tables can offer protection only if a process cannot alter their content. Therefore, a user process is restricted to only touching its address space under the user mode. Hardware requires the CPU to be in the kernel mode to modify the address translation tables.

Table 16.3 Three Illusions for Physical and Virtual Memory

Physical Memory	Virtual Memory
No protection	Each program isolated from all others and from the OS
Limited size	Illusion of infinite memory
Sharing visible to programs	Each program cannot tell whether memory is shared

A CPU can change from kernel to user mode when starting a program, or vice versa through either voluntary or involuntary mechanisms. The voluntary mechanism uses system calls, where a user application asks the operating system to do something on its behalf. A system call passes arguments to an operating system, either through registers or copying from the user memory to the kernel memory. A CPU can also be switched from user to kernel mode involuntarily by hardware interrupts (e.g., I/O) and program exceptions (e.g., segmentation fault).

On system calls, interrupts, or exceptions, hardware atomically performs the following steps: (1) sets the CPU to kernel mode; (2) saves the current program counter; (3) jumps to the handler in the kernel (the handler saves old register values).

Unlike threads, context switching among processes also involves saving and restoring pointers to translation tables. To resume execution, the kernel reloads old register values, sets the CPU to user mode, and jumps to the old program counter.

Communication among address spaces is required in this operation. Since address spaces do not share memory, processes have to perform inter-process communication (IPC) through the kernel, which can allow bugs to propagate from one program to another.

Protection by hardware can be prohibitively slow, since applications have to be structured to run in separate address spaces to achieve fault isolation. In the case of complex applications built by multiple vendors, it may be desirable for two different programs to run in the same address space, with guarantees that they cannot trash each other's code or data. Strong typing and software fault isolation are used to ensure this.

(a) Protection via strong typing

If a programming language disallows the misuse of data structures, a program may trash another, even in the same address space. With some object-oriented programming, programs can be downloaded over the net and run safely because the language, compiler, and run-time system prevents the program from doing bad things (e.g., make system calls). For example, Java defines a separate virtual machine layer, so a Java program can run on different hardware and operating systems. The downside of this protection mechanism is the requirement to learn a new language.

(b) Protection via software fault isolation

A language-independent approach is to have compilers generate code that is proven safe (e.g., a pointer cannot reference illegal addresses). For example, a pointer can be checked before it is used in some applications.

16.5.4 Memory access control

Dealing with race conditions is also one of the difficult aspects of memory management. To manage memory access requests coming from the system, a scheduler is necessary in the application layer or in the kernel, in addition to the MMU as a hardware manager. The most common way of protecting data from concurrent access by the memory access request scheduler is memory request contention. The semantics and methodologies of memory access request contention should be the same as for I/O request contention. Section 16.3.3 can be referred to for this topic.

16.6 EVENT BROKERS

An event is a case or a condition that requires the services of the microprocessor to handle it. Internal events are triggered by a running program and may result from software errors, such as a protection violation, or from execution exception, such as a page fault. External events are created by sources outside the execution of the current program and may include I/O events such as a disk request, timer-generated events, or even a message arrival in a message-passing multiprocessor system.

The response of a microprocessor to an event may be precise or imprecise, and may be concurrent or sequential. The response to an event is precise if proper handler execution ensures correct program behavior. This is guaranteed if the particular instruction causing the event (the faulting instruction) is identified, and any instructions that need the result of the faulting instruction are not issued until the handler has generated this result.

This definition of precise event handling enables us to use multitasking or multithreading to implement concurrent event handling; the event handler and the faulting program run concurrently in different task or thread contexts. This is in contrast to sequential event handling in which an event interrupts the faulting program, brings it to a sequentially consistent state, runs the handler to completion, and resumes the faulting program. Both concurrent and sequential event handling achieve the same end; all instructions get correct input operands, but do so through different means.

In this section, concurrent event handling with a multitasking or multithreading architecture will be highlighted, since it has several advantages over sequential event handling.

16.6.1 Event notification service

An event notification service (ENS) connects providers of information and interested clients. The ENS informs the clients about the occurrence of events from the providers.

Events can be, for instance, changes in existing objects or the creation of new objects such as a new measurement coming from a meter in a control system. The service learns about them by means of event messages, compiled either by the providers (push) or by the service actively observing the providers (pull). Clients define their interest in certain events by means of personal profiles. Profile creation is based on the profile definition language of the service. The information about observed events is filtered according to these profiles, and notifications are sent to the interested clients.

Figure 16.17 depicts the dataflow in the high-level architecture of an ENS. Keep in mind that the dataflow is independent of the delivery mode, such as push or pull. The tasks of an ENS can be subdivided into four steps. Firstly, the observable event (message) types are determined and advertised to the clients. This advertisement can be implicit, for example, provided by the client interface or due to information the client has about the service. Secondly, the clients' profiles have to be defined through a client interface; they are stored within the service. Thirdly, the occurring events have to be observed and filtered by the ENS. Before creating notifications, the event information is combined to detect composite events. The messages can be buffered to enable more efficient notification (e.g., by merging several messages into one notification). Finally the clients have to be notified, immediately or according to a given schedule.

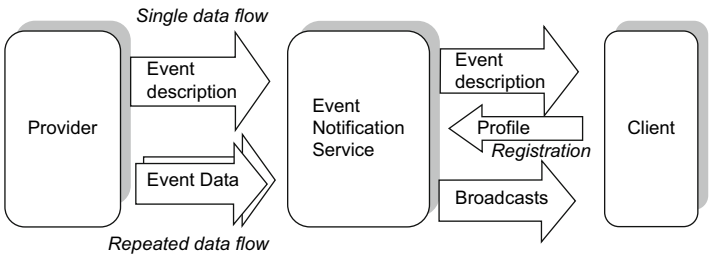


FIGURE 16.17

The dataflow of an event notification service.

In object-oriented programming designs, the following considerations regarding the ENS can be very helpful.

(1) In programs, the event and its data can be written as abstract objects in the operating system code each can be an instance of the abstract event object. Event-data should be linked to an event. Each of these event objects includes its own trigger list, each node of which can be a client object to this event. Each client related to the same event should be registered into the trigger list of the corresponding event, either dynamically in run-time or initially when the system powers on. Data structures such as a hash-table can be chosen to maintain the event-client diagram in programs.

(2) The client object as nodes in the trigger list should include the handlers of the linked event, which can be routines, functions, or methods. When an event occurs while a program is running, the program context should go to the trigger list of this event immediately. Each node of this trigger list is drawn and the corresponding event handlers are run. This is called broadcast semantics.

(3) In multitask or multithread programming architecture, each of these routines can be implemented by one or more tasks or threads. Concurrent event handling requires synchronization between the handling routines. Although there are several ways of pursuing this synchronization, the context switch of tasks or threads in the kernel of a microprocessor chip seems essential for this purpose. We can also use software interrupts to deal with event handling routines; the occurrence of each event creates a software interrupt that leads the context to the handling routines. When the handling routines are complete, the context restores to the original process. However, for distributed control systems, message communications are the basic method of managing synchronization between the handling routines in different microprocessor chipsets.

16.6.2 Event triggers and broadcasts

Event triggers (or flags) are operations linked to types and associated with events. When the specified event occurs on an object of the linked type, the trigger operation will be invoked automatically. A trigger specifies its name along with the event that will trigger its execution. Four types of events may be specified: create, update, delete, and commit. A create event triggers execution after creation. Update and delete events trigger execution before any changes are made to the actual stored object. A commit event triggers execution before the commit is performed. All triggers and events refer only to

that part of the object associated with the type in question. Thus, so addressing an object with an additional type will cause a new type part to be created and hence any create triggers associated with the acquired type to be invoked.

The event broadcast is responsible for sending a notification to all client applications that have previously registered into this event including the notification identifier of the notifying session server.

When broadcast semantics is used to signal the occurrence of the events, a useful programming technique is to put the broadcasts in a rule that is used to wake up applications that need to pay attention to something. In multitasking (or multithreading) operating systems, a reactive task (or reactive thread) mechanism is often used to wake up the applications that need to handle the event. The client will get back an event notification, just like all the other listening sessions. Depending on the application logic, this could result in useless work.

16.6.3 Event handling routines

An event handling routine is required to handle a given type of event. Here, the sources of the events are determined by reading event flags (or event triggers). These should then be reset and one event should be handled at a time. The handling of events continues in this fashion until all event flags are found to be inactive, and then the event routine is left.

It is recommended to use a trap mechanism to execute event handling routines, especially at the hardware level. This involves a set of event handling routines executed on behalf of user programs by the operating system, in which the trap instruction starts when a user program wishes the operating system to execute a specific routine, and then return control to itself. The event handling routine must have a mechanism for returning control from the operating system to the user program after completion. Similarly to interrupt handling mechanisms, this trap instruction causes the event service to execute the handling routine by changing the microprocessor to the starting address of the relevant routine on the basis of its trap vector. The interrupt vector is where the interrupt handlers are stored in an array of pointers. It also provides a way to get back to the program that initiated the trap instruction, referred to as a linkage.

After an event is handled, it resets the event flag first and then handles the particular event. If a new event from the same source arrives before the flag is reset and the previous event has been handled, this new event might be lost when the flag is reset. Thus, the chances of losing events are decreased when the flag is reset immediately.

However, if using a trap mechanism, event flags should be read again after handling the event, because the microprocessor trap may not exit from the event routine when there are pending events. While a previous event is being handled, a new event may have arrived (of the same type or a different type). By reading the event flags after each handled event, the user or programmer can easily implement an event priority scheme.

16.7 MESSAGE QUEUE

16.7.1 Message passing

An operating system provides inter-process communication (IPC) to allow processes to exchange information, which is useful for creating cooperating processes. The most popular form of IPC

involves message passing. A process may send information to a channel (or port), from which another process may receive it. The sending and receiving processes can be on the same or different computers connected through a communication medium.

One reason for the popularity of message passing is its ability to support client server interaction. A server is a process that offers a set of services to client processes, which are invoked in response to messages from the clients and results are returned in messages to them. We are particularly interested in servers that offer operating system services. With such servers, part of the operating system functionality can be transferred from the kernel to utility processes, for instance, file management can be handled by a file server, which offers services such as open, read, write, and seek.

There are several issues involved in message passing. We discuss some of these below.

(1) Reliability and order

Messages sent between computers can fail to arrive or can be garbled because of noise and contention for the communication line. There are techniques to increase the reliability of data transfer, but they cost both extra space (longer messages to increase redundancy, more code to check the messages) and time. Message-passing techniques can be distinguished by the reliability with which they deliver messages.

Another issue is whether messages sent to a channel are received in the order in which they are sent. Differential buffering delays and routings in a network environment can place messages out of order. It takes extra effort (in the form of sequence number, and more generally, time stamps) to ensure order.

(2) Access

An important issue is how many readers and writers can exchange information through a channel. Different approaches impose various restrictions on the access to channels. A bound channel is the most restrictive: there may be only one reader and writer. At the other extreme, a free channel allows any number of readers and writers. These are suitable for programming client-server interactions based on a family of servers providing a common service. This is associated with a single channel; clients send their service requests to this channel and servers providing the requested service receive service requests from the channel. Unfortunately, implementing free channels can be quite costly if in-order messages are to be supported.

The message queue associated with the channel is kept at a site which, in general, is remote to both sender and receiver. Thus both send and receive requests-result in messages being sent to this site the former put messages in this queue and the latter request messages from it. Between these two extremes are input channels and output channels. An input channel has only one reader but any number of writers modelling the fairly typical many-client, one-server situation. They are easy to implement since all receivers that designate a channel occur in the same process. Thus, the message queue associated with a channel can be kept with the receiver. Output channels, in contrast, allow any number of readers but only one writer. They are easier to implement than free channels since the message queue can be kept with the sender, but they are not much used since one-client and many-server situations are very unusual.

Several applications can use more than one kind of channel. For instance, a client can enclose a bound channel in a request message to the input port of a file server. This bound channel can represent the opened file, and subsequent read and write requests can be directed to this channel.

(3) Synchronous and asynchronous

The send, receive, and reply operations may be synchronous or asynchronous. A synchronous operation blocks a process till the operation completes, whereas an asynchronous operation is non-blocking and only initiates the operation. The caller could discover completion by some other mechanism. Note that both synchronous and asynchronous imply blocking and not blocking but not vice versa, that is, not every blocking operation is synchronous and not every non-blocking operation is asynchronous. For instance, a send that blocks till the receiver machine has received the message is blocking but not synchronous, since the receiver process may not have received it. These definitions of both synchronous and asynchronous operations are similar but not identical to the ones given in textbooks, which tend to equate synchronous with blocking.

Asynchronous message passing allows more parallelism. Since a process does not block, it can do some computation while the message is in transit. In the receive case, a process can express its interest in receiving messages on multiple channels simultaneously. In a synchronous system, such parallelism can be achieved by forking a separate process for each concurrent operation, but this approach incurs the cost of extra process management.

Asynchronous message passing introduces several problems. What happens if a message cannot be delivered? The sender may not wait for delivery of the message, and thus never hear about the error. Similarly, a mechanism is needed to notify an asynchronous receiver that a message has arrived. The operation invoker could learn about completion or errors by polling, getting a software interrupt, or by waiting explicitly for completion later using a special synchronous wait call. An asynchronous operation needs to return a call ID if the application needs to be later notified about the operation. At notification time, this ID would be placed in some global location or passed as an argument to a handler or wait call.

Another problem related to asynchronous message passing has to do with buffering. If messages sent asynchronously are buffered in a space managed by the operating system, then a process may fill this space by flooding the system with a large number of messages.

16.7.2 Message queue types

Message queues allow one or more processes to write messages, which will be read by one or more reading processes. Most operating systems or kernels maintain a list of message queues; a vector keeps the messages, each element of which points to a data structure that fully describes the message queue. When message queues are created, a new data structure is allocated from system memory and inserted into the vector.

Figure 16.18 displays a possible design of system message queue. In Figure 16.18, each “msqid_ds” data structure contains an “ipc_” data structure and pointers to the messages entered onto this queue. In addition, it keeps queue modification times such as the last time that this queue was written to and so on. The “msqid_ds” also contains two wait queues, one for the writers to the queue and one for the readers of the message queue.

Each time a process attempts to write a message to the write queue, its effective user and group identifiers are compared with the mode in this queue’s “ipc_” data structure. If the process can write to the queue then the message may be copied from the process’s address space into the “a_msg” data structure and put at the end of this message queue. Each message is tagged with an application-specific

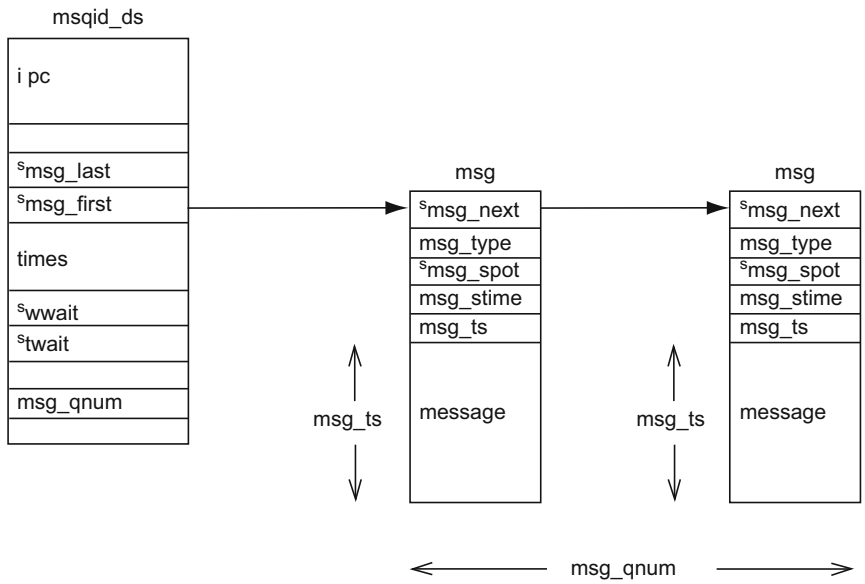


FIGURE 16.18

A design for system message queues.

type, agreed between the cooperating processes. However, there may be no room for the message as the operating system may restrict the number and length of messages that can be written. In this case, the process will be added to this message queue's write wait queue and the scheduler will be called to select a new process to run. It will be woken up when one or more messages have been read from this message queue.

Reading from the queue is a similar process. Again, the process's access rights to the write queue are checked. A reading process may choose to either get the first message in the queue regardless of its type, or select messages of a particular type. If no messages match these criteria, the reading process will be added to the message queue's read wait queue and the scheduler run. When a new message is written to the queue this process will be woken up and run again.

16.7.3 Pipes

In implementations of IPC, one solution to some of the buffering problems of asynchronous sending is to provide an intermediate degree of synchrony. We can treat the set of message buffers as a "traditional bounded buffer" that blocks the sending process when there are no more buffers available. That is exactly the kind of message passing supported by pipes.

Pipes also allow the output of one process to become the input of another. A pipe is like a file opened for reading and writing. They are constructed by the service call pipe, which opens a new pipe and returns two descriptors for it, one for reading and another for writing. Reading from a pipe advances the read buffer, and writing to it advances the write buffer. The operating system

may only wish to buffer a limited amount of data for each pipe, so an attempt to write to a full pipe may block the writer. Similarly, an attempt to read from an empty buffer will block the reader.

Though a pipe may have several readers and writers, it is really intended for one reader and writer. Pipes are used to unify both the input and output mechanisms and IPC. Processes expect that they will have two descriptors when they start, one called standard input and another called standard output. Typically, the first is a descriptor for the terminal open for input, and the second is a similar descriptor for output. However, the command interpreter, which starts most processes, can arrange for these descriptors to be different. If the standard output descriptor happens to be a file descriptor, the output of the process will go to the file, and not to the terminal. Similarly, the command interpreter can arrange for the standard output of one process to be one end of a pipe and for the other end of the pipe to be standard input for a second process. Thus, a listing program can be piped to a sorting program, which in turn directs its output to a file.

Conceptually, a pipe can be thought of much like a hose-pipe, in that it is a conduit where we pour data in at one end and they flow out at the other. A pipe looks a lot like a file, in that it is treated as a sequential data stream. Unlike a file, there is no physical storage of data when we close a pipe; anything that was written in one end but not read out from the other end will be lost.

The use of pipes as conduits between processes is shown in the diagram given in Figure 16.19. Here we see two processes that are called parent and child, for reasons that will become apparent shortly. The parent can write to Pipe A and read from Pipe B. The child can read from Pipe A and write to Pipe B.

A pipe can be implemented using two file data structures that both point at the same temporary inode, which itself points at a physical page within memory. Figure 16.20 shows that each file data structure contains pointers to two different file operation routine vectors: one for writing to the pipe, the other for reading from the pipe. This hides the underlying differences from generic system calls that read and write to ordinary files. As the writing process writes to the pipe, bytes are copied into the shared data page and when the reading process reads from the pipe, bytes are copied from its shared data page. The operating system must synchronize access to the pipe. It must make sure that the reader and the writer of the pipe are in step and to do this it uses locks, wait queues, and signals.

When the writer wants to write to the pipe it uses the standard write library functions. These all pass file descriptors that are indices into the process's set of file data structures, each one representing an open file, or, as in this case, an open pipe. The operating system call uses the write routine pointed at by

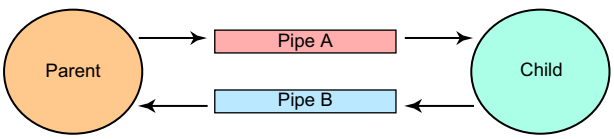


FIGURE 16.19

Pipes used by parent and child processes for IPC.

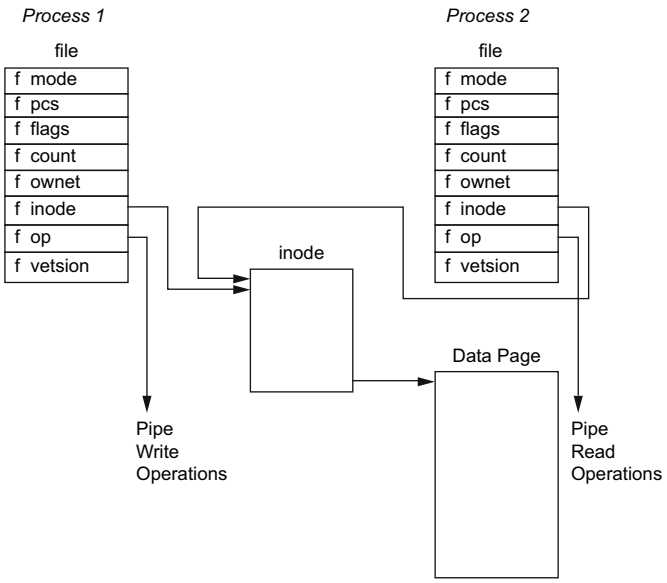


FIGURE 16.20

The pipes between process 1 and process 2.

the file data structure describing this pipe, which in turn uses information held in the inode representing the pipe to manage the write request.

If there is enough room to write all of the bytes into the pipe then, so long as the pipe is not locked by its reader, the operating system locks it for the writer and copies the bytes to be written from the process's address space into the shared data page. If the pipe is locked by the reader, or if there is not enough room for the data, then the current process is made to sleep in the pipe inode's wait queue and the scheduler is called so that another process can run. It is interruptible, so it can receive signals and will be woken by the reader when there is enough room for the write data, or when the pipe is unlocked. When the data have been written, the pipe's inode is unlocked and any waiting readers sleeping in the wait queue of the inode will themselves be woken up. Reading data from the pipe is a very similar process to writing to it.

Some operating systems also support named pipes, also known as FIFOs because pipes operate on a first in, first out principle; the first data written into the pipe are the first read from it. Unlike pipes, FIFOs are not temporary objects; they are entities in the file system and can be created using system commands. Processes are free to use a FIFO so long as they have appropriate access rights to it. The way that FIFOs are opened is a little different from pipes. A pipe (its two file data structures, its inode, and the shared data page) is created in one go whereas a FIFO already exists and is opened and closed by its users. It must handle readers opening the FIFO before writers open it, as well as readers reading before any writers have written to it. That aside, FIFOs are handled almost exactly the same way as pipes and they use the same data structures and operations.

16.8 SEMAPHORES

Semaphores are used to control access to shared resources by processes, and can be named or unnamed. Named semaphores provide access to a resource shared between multiple processes, while unnamed semaphores provide multiple accesses to a resource within a single process or shared between related processes. Some semaphore functions are specific to either named or unnamed semaphores.

A semaphore is an integer variable taking on values from 0 to a predefined maximum, each being associated with a queue for process suspension. The order of process activation from the queue must be fair. Two atomic operations that are indivisible and uninterruptible are defined for a semaphore:

(a) WAIT: decrease the counter by one; if it becomes negative, block the process and enter this process's ID in the waiting processes queue.
(b) SIGNAL: increase the semaphore by one; if it is still negative, unblock the first process of the waiting processes queue, removing this process's ID from the queue itself.

In its simplest form, a semaphore is a location in memory whose value can be tested and set by more than one process. The test and set operation is, so far as each process is concerned, atomic; once started nothing can stop it. The result of this operation is the addition of the current value of the semaphore to the set value, which can be positive or negative. Depending on the result of the operation, one process may have to sleep until the semaphore's value is changed by another.

Say you had many cooperating processes reading records from and writing records to a single data file. You would want that file access to be strictly coordinated. A semaphore with an initial value of 1 could be used, and, around the file operating code, two semaphore operations could be implemented, the first to test and decrement the semaphore's value and the second to test and increment it. The first process to access the file would try to decrement the semaphore's value and it would succeed, the semaphore's value now being 0. This process can now go ahead and use the data file. If another process wishing to use the file now tries to decrement the semaphore's value, it would fail as the result would be -1. That process will be suspended until the first process has finished with the data file. When the first process has finished with the data file it will increment the semaphore's value, making it 1 again. Now the waiting process can be woken and this time its attempt to increment the semaphore will succeed.

If all of the semaphore operations have succeeded and the current process does not need to be suspended, the system's operating system goes ahead and applies the operations to the appropriate members of the semaphore array. Now it must check any waiting or suspended processes that may now apply for their semaphore operations, looking at each member of the operations pending queue in turn, testing to see whether the semaphore operations will succeed this time. If they will, then it removes the data structure representing this process from the operations pending list and applies the semaphore operations to the semaphore array. It wakes up the sleeping process, making it available to be restarted the next time the scheduler runs. The operating system keeps looking through the pending processes queue from the start until there is a pass where no semaphore operations can be applied and so no more processes can be woken.

16.8.1 Semaphore depth and priority

Semaphores are global entities and are not associated with any particular process. In this sense, semaphores have no owners, making it impossible to track ownership for any purpose, including error recovery. Semaphore protection works only if all the processes using the shared resource cooperate by waiting for it when it is unavailable and incrementing the semaphore value when relinquishing the resource. Since semaphores lack owners, there is no way to determine whether one of the cooperating processes has become uncooperative. Applications using semaphores must carefully detail cooperative tasks. All of the processes that share a resource must agree on which semaphore controls the resource.

There is a problem with semaphores, called deadlocks, which occur when one process has altered the semaphore's value as it enters a critical region but then fails to leave this region because it crashed or was killed. Maintaining lists of adjustments to the semaphore arrays offers some protection. The idea is that when these adjustments are applied, the semaphores will be put back to the state that they were in before the process set of semaphore operations was applied.

There is another problem, known as process priority inversion, which is a form of indefinite postponement common in multitasking, preemptive executives with shared resources. It occurs when a high-priority task requests access to a shared resource which is currently allocated to a low-priority task. The high-priority task must block until the low-priority task releases the resource. This problem is exacerbated when the low-priority task is prevented from executing by one or more medium-priority tasks. Because the low-priority task is not executing, it cannot complete its interaction with the resource and release it. The high-priority task is effectively prevented from executing by lower-priority tasks.

Priority inheritance is an algorithm that calls for the lowest-priority task holding a resource to have its priority increased to that of the highest-priority task currently waiting for that resource. Each time a task blocks attempting to obtain the resource, the task holding the resource may have its priority increased. Some kernels support priority inheritance for local, binary semaphores that use the priority task wait queue blocking discipline. When a task is of higher priority than the task holding the semaphore blocks, the priority of the task holding the semaphore is increased to that of the blocking task. When the binary semaphore is released (i.e., not for a nested release), the holder's priority is restored to its original value. The implementation of the priority inheritance algorithm takes into account the scenario in which a task holds more than one binary semaphore. The holding task will execute at the priority of the higher of the highest ceiling priority or at the priority of the highest-priority task blocked waiting for any of the semaphores the task holds. Only when the task releases all of the binary semaphores it holds will its priority be restored to the normal value.

Priority ceiling is an algorithm that calls for the lower-priority task holding a resource to have its priority increased to that of the highest-priority task which will ever block waiting for that resource. This algorithm addresses the problem of priority inversion, although it avoids changing the priority of the task holding the resource. The priority ceiling algorithm will only change the priority of the task holding the resource once. The ceiling priority is set at creation time and must be the priority of the highest-priority task which will ever attempt to acquire that semaphore. Some kernels support priority ceiling for local, binary semaphores that use the priority task wait queue blocking discipline. When a task of lower priority than the ceiling priority successfully obtains the semaphore, its priority is raised to the ceiling priority. When the task holding the task completely releases the binary semaphore (i.e., not for a nested release), the holder's priority is restored to its previous value.

Identifying the highest-priority task that will attempt to obtain a particular semaphore can be a difficult task in a large, complicated system. Although the priority ceiling algorithm is more efficient than the priority inheritance algorithm with respect to the maximum number of task priority changes that may occur while a task holds a particular semaphore, the latter is more forgiving, in that it does not require this information. The implementation of the priority ceiling algorithm takes into account the scenario where a task holds more than one binary semaphore. The holding task will execute at the priority of the higher of the highest ceiling priority or at the priority of the highest-priority task blocked waiting for any of the semaphores the task holds. Only when the task releases all of the binary semaphores it holds will its priority be restored to its normal value.

16.8.2 Semaphore acquire, release and shutdown

A semaphore can be viewed as a protected variable whose value can be modified only by methods for creating, obtaining, and releasing a semaphore. Many kernels or operating systems support both binary and counting semaphores. A binary semaphore is restricted to values of zero or one, while a counting semaphore can assume any non-negative integer value.

A binary semaphore can be used to control access to a single resource, in particular, to enforce mutual exclusion for a critical section of code. To do this, the semaphore would be created with an initial value of one to indicate that no task is executing the critical section of code. On entry to the critical section, a task must issue the method for obtaining a semaphore to block other tasks from entering. On exit from the critical section, the task must issue the method for releasing the semaphore, to allow access for another task.

A counting semaphore can be used to control access to a pool of two or more resources. For example, access to three printers could be administered by a semaphore that is created with an initial count of three. When a task requires access to one of the printers, it issues the method for obtaining a semaphore. If a printer is not currently available, the task can wait for a printer to become available or return immediately. When the task has completed printing, it should issue the method for releasing a semaphore to allow other tasks access to the printer.

Task synchronization may be achieved by creating a semaphore with an initial count of zero. One task waits for the arrival of another task by issuing a method that obtains a semaphore when it reaches a synchronization point. The other task performs a corresponding semaphore release operation when it reaches its synchronization point, thus unblocking the pending task.

(1) Creating a semaphore

This method creates a binary or counting semaphore with a user-specified name as well as an initial count. If a binary semaphore is created with a count of zero (0) to indicate that it has been allocated, then the task that creates the semaphore is considered to be its the current holder. At create time, the method for ordering tasks waiting in the semaphore's task wait queue (by FIFO or task priority) is specified. In addition, the priority inheritance or priority ceiling algorithm may be selected for local, binary semaphores that use the priority task wait queue blocking discipline. If the priority ceiling algorithm is selected, then the highest priority of any task that will attempt to obtain this semaphore must be specified.

(2) Obtaining semaphore IDs

When a semaphore is created, the kernel generates a unique semaphore ID and assigns it to the created semaphore by either of the two methods. First, after the semaphore is invoked by means of the semaphore creation method, the semaphore ID is stored in a user-provided location. Second, the semaphore's ID may be obtained later using the semaphore identify method. The semaphore's ID is used by other semaphore manager methods to access this semaphore.

(3) Acquiring a semaphore

A simplified version of this method can be described as follows: if the semaphore's count is greater than zero then decrement the semaphore's count, or else wait for release of the semaphore and return "successful".

When the semaphore cannot be immediately acquired, one of the following situations applies. By default, the calling task will wait forever to acquire the semaphore. If the task waits to acquire the semaphore, then it is placed in the semaphore's task wait queue in either FIFO or task priority order. If the task blocked waiting for a binary semaphore is using priority inheritance and the task's priority is greater than that of the task currently holding the semaphore, then the holding task will inherit the priority of the blocking task. All tasks waiting on a semaphore are returned an error code when the semaphore is deleted.

When a task successfully obtains a semaphore using priority ceiling and the priority ceiling for this semaphore is greater than that of the holder, then the holder's priority will be elevated.

(4) Releasing a semaphore

A simplified version of the semaphore release method is as follows: if no tasks are waiting for this semaphore then increment the semaphore's count, or else assign the semaphore to a waiting task and return "successful". If this is the outermost release of a binary semaphore that uses priority inheritance or priority ceiling and the task does not currently hold any other binary semaphores, then the task performing the semaphore release will have its priority restored to its normal value.

(5) Deleting a semaphore

The semaphore delete method removes a semaphore from the system and frees its control block. A semaphore can be deleted by any local task that knows its ID. As a result of this directive, all tasks blocked waiting to acquire the semaphore will be readied and returned a status code that indicates that the semaphore was deleted. Any subsequent references to the semaphore's name and ID are invalid.

16.8.3 Condition and locker

The paragraphs below present a series of basic synchronization problems including serialization and mutual exclusion, and show some ways of using semaphores to solve them.

(1) Signaling

Possibly the simplest use for a semaphore is signaling, which means that one thread sends a signal to another to indicate that something has happened. Signaling makes it possible to guarantee that

a section of code in one thread will run before a section of code in another; in other words, it solves the serialization problem.

Assume that we have a semaphore named "sem" with initial value 0, and the threads A and B have shared access to it as given in Figure 16.21. The word "statement" represents an arbitrary program statement. To make the example concrete, imagine that "a1" reads a line from a file, and "b1" displays the line on the screen. The semaphore in this program guarantees that thread A has completed a1 before thread B begins b1. Here is how it works: if thread B gets to the wait statement first, it will find the initial value, zero, and it will block. Then when thread A signals, thread B proceeds. Similarly, if thread A gets to the signal first then the value of the semaphore will be incremented, and when thread B gets to the wait, it will proceed immediately. Either way, the order of a1 and b1 is guaranteed.

This use of semaphores is the basis of the names signal and wait, and in this case the names are conveniently mnemonic. Unfortunately, we will see other cases where the names are less helpful. Speaking of meaningful names, "sem" is not one. When possible, it is a good idea to give a semaphore a name that indicates what it represents. In this case, a name such as a1_Done might be good, where a1_Done = 0 means that a1 has not executed and a1_Done = 1 means it has.

(2) Mutex

A second common use for semaphores is to enforce mutual exclusion. We have already seen one use for this, when controlling concurrent access to shared variables. The mutex guarantees that only one thread accesses the shared variable at a time.

A mutex is like a token that passes from one thread to another, allowing one thread at a time to proceed. For example, in "The Lord of the Flies" a group of children use a conch as a mutex. To speak, you have to hold the conch. As long as only one child holds the conch, only one can speak. Similarly, in order for a thread to access a shared variable, it has to "get" the mutex; when it is finished, it "releases" the mutex. Only one thread can hold the mutex at a time.

Create a semaphore named mutex that is initialized to 1. A value of "one" (1) means that a thread may precede and may access the shared variable. A value of "zero" (0) means that it has to wait for another thread to release the mutex. A code segment for performing mutex by semaphore is given in Figure 16.22.

Since mutex is initially 1, whichever thread gets to the wait first will be able to proceed immediately. Of course, the act of waiting on the semaphore has the effect of decrementing it, so the second thread to arrive will have to wait until the first signals.

In this example, both threads are running the same code. This is sometimes called a symmetric solution. If the threads have to run different code, the solution is asymmetric. Symmetric solutions are often easier to generalize. In this case, the mutex solution can handle any number of concurrent threads without modification. As long as every thread waits before performing an update and signals

Thread A			Thread B		
1	statement	a1	1	sem.wait()	
2	sem.signal()		2	statement	b1

FIGURE 16.21

Signaling by semaphore.

Thread A	Thread B

mutex.wait()
  # critical section
  count = count + 1
mutex.signal()
	mutex.wait()
	  # critical section
	  count = count + 1
	mutex.signal()

FIGURE 16.22

Mutex by semaphore.

afterwards, then no two threads can access the count concurrently. Often the code that needs to be protected is called the critical section, to which it is critically important to prevent concurrent access.

In the tradition of computer science and mixed metaphors, there are several other ways people sometimes talk about mutexes. In the metaphor we have been using so far, the mutex is a token that is passed from one thread to another. In an alternative metaphor, we think of the critical section as a room, and only one thread is allowed to be in the room at a time. In this metaphor, mutexes are called locks, and a thread is said to lock the mutex before entering and unlock it while exiting. Occasionally, though, people mix the metaphors and talk about "getting" or "releasing" a lock, which does not make much sense. Both metaphors are both potentially useful and potentially misleading. As you work on the next problem, try to figure out both ways of thinking and see which one leads you to a solution.

(3) Multiplex

Generalize the previous solution so that it allows multiple threads to run in the critical section at the same time, but it enforces an upper limit on the number of concurrent threads. In other words, no more than n threads can run in the critical section at the same time.

This pattern is called a multiplex. In real life, the multiplex problem occurs at busy nightclubs where there are a maximum number of people allowed in the building at a time, either to maintain fire safety or to create the illusion of exclusivity. At such places a bouncer usually enforces the synchronization constraint by keeping track of the number of people inside and barring arrivals when the room is at capacity. Then, whenever one person leaves another is allowed to enter. Enforcing this constraint with semaphores may sound difficult, but it is almost trivial (Figure 16.23).

To allow multiple threads to run in the critical section, just initialize the mutex to n, which is the maximum number of threads that should be allowed. At any time, the value of the semaphore

Multiplex solution
1 multiplex.wait() 2 critical section 3 multiplex.signal()

FIGURE 16.23

Multiplex by semaphore.

represents the number of additional threads that may enter. If the value is zero, then the next thread will block until one of the threads inside exits and signals. When all threads have exited the value of the semaphore is restored to n.

Since the solution is symmetric, it is conventional to show only one copy of the code, but you should imagine multiple copies of the code running concurrently in multiple threads.

What happens if the critical section is occupied and more than one thread arrives? Of course, what we want is for all the arrivals to wait. This solution does exactly that. Each time an arrival joins the queue, the semaphore is decremented, so that the value of the semaphore (negated) represents the number of threads in the queue. When a thread leaves, it signals the semaphore, incrementing its value and allowing one of the waiting threads to proceed.

Thinking again of metaphors, in this case it is useful to think of the semaphore as a set of tokens (rather than a lock). As each thread invokes wait, it picks up one of the tokens; when it invokes a signal it releases one. Only a thread that holds a token can enter the room, and if none are available when a thread arrives, it waits until another thread releases one.

In real life, ticket windows sometimes use a system like this. They hand out tokens (sometimes poker chips) to customers in line. Each token allows the holder to buy a ticket.

(4) Barrier

The barrier method is hinted at by presenting the variables used in the solution and examining their roles in Figure 16.24.

Finally, here in Figure 16.25 is a working barrier. The only change from signaling is the addition of another signal after waiting at the barrier. Now as each thread passes, it signals the semaphore so that the next thread can pass. This pattern, a wait and a signal in rapid succession, occurs often enough that it has a name; it is called a turnstile, because it allows one thread to pass at a time, and it can be locked to bar all threads.

In its initial state (zero), the turnstile is locked. The n-th thread unlocks it and then all n threads go through.

16.9 TIMER

Most computer control systems include electronic timers, which are usually just digital counters that are set to a number by software, and then count down to zero. When they reach zero, they interrupt the microprocessor or CPU, which triggers interrupt handling. Another common form of timer is a number

Barrier hint

```
1  int n                  # the number of threads
2  int count = 0
3  Semaphore mutex = 1
4  Semaphore barrier = 0
```

FIGURE 16.24

Barrier by semaphore (1).

Barrier solution

```
1
2     mutex.wait()
3           count = count + 1
4     mutex.signal()
5
6     if count == n: barrier.signal()
7
8     barrier.wait()
9     barrier.signal()
10
11    critical point
```

FIGURE 16.25

Barrier by semaphore (2).

that is compared to a counter. This is somewhat harder to program, but can be used to measure events or to control motors (using a digital electronic amplifier to perform pulse-width modulation). Embedded systems often use a hardware timer to implement a list of software timers. Basically, the hardware timer is set to expire at the time the next software timer comes up. The hardware timer's interrupt software handles the housekeeping of notifying the rest of the software, finding the next software timer to expire, and resetting the hardware timer to the next software timer's expiration.

16.9.1 Kernel timers

Whenever you need to schedule an action to happen later, without blocking the current process until that time arrives, kernel timers are the tool for you. These timers are used to schedule execution of a function at a particular time in the future, based on the clock tick, and can be used for a variety of tasks; for example, polling a device by checking its state at regular intervals when the hardware cannot fire interrupts. Other typical uses of kernel timers are turning off the floppy motor or finishing another lengthy shut down operation. Finally, the kernel itself uses timers in several situations, including the implementation of schedule timeout.

A kernel timer is a data structure that instructs the kernel to execute a function with an argument at a given time, all user-defined. The functions almost certainly do not run while the process that registered them is executing. When a timer runs, however, the process that scheduled it could be asleep, executing on a different microprocessor, or quite possibly have exited altogether.

In fact, kernel timers are run as the result of a software interrupt. When running in this sort of atomic context, your code is subject to a number of constraints. Timer functions must be atomic in all ways, but there are some additional issues brought about by the lack of a process context. Repetition is called for because the rules for atomic contexts must be followed assiduously, or the system will find itself in deep trouble.

One other important feature of kernel timers is that a task can reregister itself to run again at a later time. This is possible because each timer list structure is unlinked from the list of active timers before being run and can, therefore, be immediately relinked elsewhere. Although rescheduling the same task

over and over might appear to be a pointless operation, it is sometimes useful, for example, in the polling of devices. Therefore, a timer that reregisters itself always runs on the same CPU.

An important feature of timers is that they are a potential source of race conditions, even on single-processor systems because they are asynchronous with other code. Therefore, any data structures accessed by the timer function should be protected from concurrent access, either by being atomic types or by using spin locks.

The implementation of the kernel timers has been designed to meet the following requirements and assumptions: (1) timer management must be as lightweight as possible; (2) the design should scale well as the number of active timers increases; (3) most timers expire within a few seconds or minutes at most, while timers with long delays are pretty rare; (4) a timer should run on the same CPU that registered it.

The solution devised by kernel developers is based on a per-CPU data structure. The "Timer list" structure includes a pointer to that data structure in its base field. If base is null, the timer is not scheduled to run; otherwise, the pointer tells which data structure (and, therefore, which CPU) runs it. Whenever kernel code registers a timer, the operation is eventually performed, which, in turn, adds the new timer to a double-linked list within a cascading table associated with the current CPU.

The cascading table works like this: if the timer expires in the next 0 255 jiffies, it is added to one of the 256 lists devoted to short-range timers using the least significant bits of the expired field. If it expires farther in the future (but before 16,384 jiffies), it is added to one of 64 lists based on bits 9 14 of the expired fields. For timers expiring even farther, the same trick is used for bits 15 20, 21 26, and 27 31. Timers with an expire field pointing still farther into the future (something that can happen only on 64-bit platforms) are hashed with a delay value of 0xffffffff, and timers that expired in the past are scheduled to run at the next timer tick. (A timer that is already expired may sometimes be registered in high-load situations, especially if you run a preemptible kernel.)

Keep in mind, however, that a kernel timer is far from perfect, as it suffers from artifacts induced by hardware interrupts, as well as other timers and other asynchronous tasks. While a timer associated with simple digital I/O can be enough for simple tasks such as running a stepper motor or other amateur electronics, it is usually not suitable for production systems in industrial environments. For such tasks, you will most likely need to resort to a real-time kernel extension.

16.9.2 Watchdog timers

(1) Working mechanism

A watchdog timer is a piece of hardware, often built into a microcontroller or CPU chipset, that can cause a microprocessor reset when it judges that the system has hung, or is no longer executing the correct sequence of program code. The hardware component of a watchdog is a counter that is set to a certain value and then counts down towards zero. It is the responsibility of the software to set the count to its original value often enough to ensure that it never reaches zero. If it does reach zero, it is assumed that the software has failed in some manner and the CPU is reset.

It is also possible to design the hardware so that a kick which occurs too soon will cause a bite, but to use such a system, very precise knowledge of the timing characteristics of the main loop of the program is required. A properly designed watchdog mechanism should, at the very least, catch events that hang the system. In electrically noisy environments, a power glitch may corrupt the program counter, stack pointer, or data in RAM, and the software would crash almost immediately,

even if the code is completely bug-free. This is exactly the sort of transient failure that watchdogs will catch.

Bugs in software can also cause the system to hang, if they lead to an infinite loop, an accidental jump out of the code area of memory, or a deadlock condition (in multitasking situations). Obviously, it is preferable to fix the root cause, rather than getting the watchdog to pick up the pieces, but in a complex embedded system it may not be possible to guarantee that there are no bugs. At least by using a watchdog you can guarantee that none of those bugs will hang the system indefinitely.

Once your watchdog has bitten, you have to decide what action to take. The hardware will usually assert the microprocessor's reset line, but other actions are also possible. For example, when the watchdog bites it may directly disable a motor, engage an interlock, or sound an alarm until the software recovers. Such actions must leave the system in a safe state if, for some reason, the system's software is unable to run at all (perhaps due to chip death) after the failure.

A microcontroller or CPU with an internal watchdog will almost always contain a status bit that gets set when a bite occurs. By examining this bit after emerging from a watchdog-induced reset, we can decide whether to continue running, switch to a fail-safe state, and/or display an error message. At the very least, you should count such events, so that a persistently errant application would not be restarted indefinitely. A reasonable approach might be to shut the system down if three watchdog bites occur in one day.

If we want the system to recover quickly, initialization after a watchdog reset should be much shorter than at power-on. On the other hand, in some systems it is better to do a full set of self-tests, since this might identify the root cause of the watchdog timeout. In terms of the outside world, the recovery may be instantaneous, and the user may not even know a reset occurred. The recovery time will be the length of the watchdog timeout plus the time it takes the system to reset and perform its initialization. How well the device recovers depends on how much persistent data it requires, and whether those data are stored regularly and read after the system resets.

(2) Sanity checks

Kicking the watchdog at a regular interval proves that the software is running. It is often a good idea to kick the watchdog only if the system passes some sanity check, as shown in Figure 16.26. Stack depth, number of buffers allocated, or the status of some mechanical component may be checked before deciding to kick the watchdog. Good design of such checks will help the watchdog to detect more errors.

One approach is to clear a number of flags before each loop is started, as shown in Figure 16.27. Each flag is set at a certain point in the loop. At the bottom of the loop the watchdog is kicked, but first the flags are checked to see that all of the important points in the loop have been visited. The multitasking or the multithreads approach is based on a similar set of sanity flags.

For a specific failure, it is often a good idea to try to record the cause (possibly in non-volatile RAM), since this may be difficult to establish after the reset. If the watchdog bite is due to a bug, then any other information you can record about the state of the system or the currently active task will be valuable when trying to diagnose the problem.

(3) Timeout interval

Any safety chain is only as good as its weakest link, and if the software policy that is used to decide when to kick the watchdog is not good, watchdog hardware can make the system less reliable. One

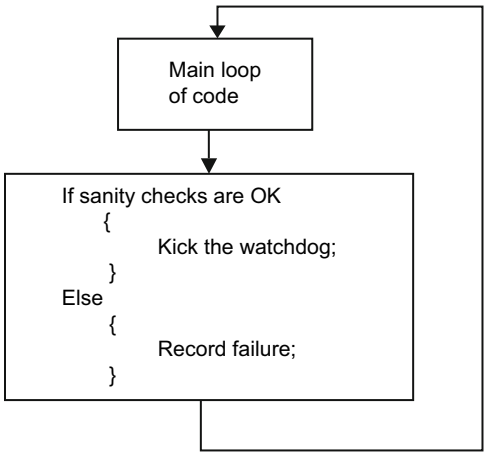


FIGURE 16.26

At the end of each execution of the main loop, the watchdog is kicked before starting again.

approach is to pick an interval that is several seconds long. This is a robust approach; some systems require fast recovery, but others only need the system not to be left in a hung state indefinitely. For these more sluggish systems, there is no need to do precise measurements of the worst-case time of the program's main loop to the nearest millisecond.

When picking the timeout one needs to consider the greatest amount of damage the device can do between the original failure and the watchdog biting. With a slowly responding system, such as a large thermal mass, it may be acceptable to wait 10s before resetting, since this will guarantee that there will be no false watchdog resets.

While on the subject of timeouts, it is worth pointing out that some watchdog circuits allow the very first timeout to be considerably longer than those following. This allows the microprocessor to initialize without having to worry about the watchdog biting.

Although the watchdog can often respond fast enough to halt mechanical systems, it offers little protection for damage that can be done by software alone. Consider an area of nonvolatile RAM that may be overwritten with rubbish data if some loop goes out of control. It is likely that overwrite would occur far faster than a watchdog could detect the fault, and for those situations some other protection, such as a checksum may be needed. The watchdog is really just one layer of protection, and should form part of a comprehensive safety network.

On some microcontrollers or CPUs, the built-in watchdog has a maximum timeout of the order of a few hundred milliseconds, obtained by multiplying the time interval in software. Say the hardware provides a 100ms timeout, but you only want to check the system for sanity every 300ms. You will have to kick the watchdog at an interval shorter than 100ms, but it will only do the sanity check every third time the kick function is called. This approach may not be suitable for a single-loop design if the main loop could take longer than 100ms to execute.

One possibility is to move the sanity check out to an interrupt. This would be called every 100ms, and would then kick the watchdog. On every third try, the interrupt function would check a flag that

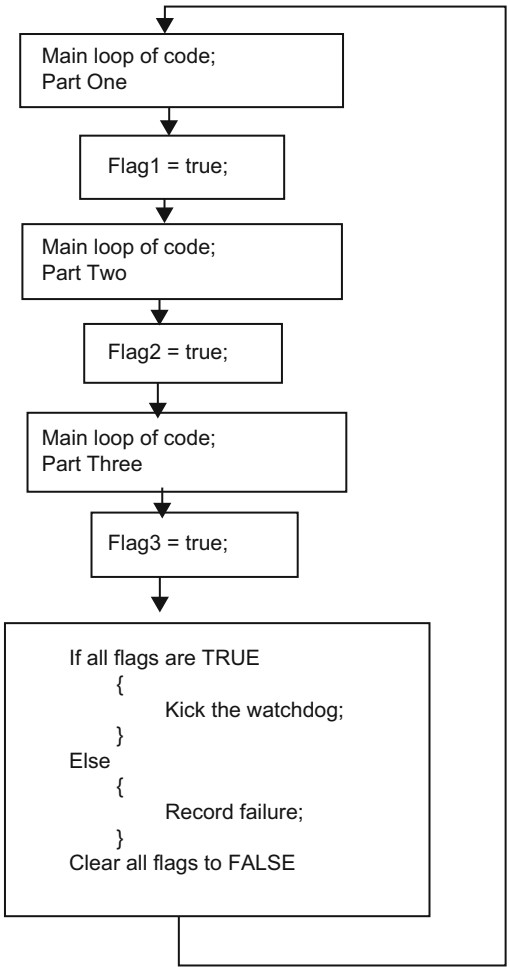


FIGURE 16.27

Use three flags to check that certain points within the main loop have been visited.

indicates that the main loop is still spinning. This flag is set at the end of the main loop, and cleared by the interrupt as soon as it has read it. If kicking the watchdog from an interrupt, it is vital to have a check on the main loop. Otherwise it is possible to get into a situation where the main loop has hung, but the interrupt continues to kick the watchdog, which never gets a chance to reset the system.

(4) Self-test

Assume that the watchdog hardware fails in such a way that it never bites. The fault would only be discovered when some failure that normally leads to a reset leads instead to a hung system. If such a failure was acceptable, the watchdog was not needed in the first place.

Many systems contain a means to disable the watchdog, such as a jumper that connects its output to the reset line. This is necessary for some test modes, and for debugging with any tool that can halt the program. If the jumper falls out, or a service engineer who removed the jumper for a test forgets to replace it, the watchdog will be rendered toothless.

The simplest way for a device to do a start-up self-test is to allow the watchdog to timeout, causing a microprocessor to be reset. To avoid looping infinitely in this way, it is necessary to distinguish the power-on case from the watchdog reset case. If the reset was due to a power-on, then perform this test, but if due to a watchdog bite, then we may already be running the test. By writing a value in RAM that will be preserved through a reset, you can check whether the reset was due to a watchdog test or to a real failure. A counter should be incremented while waiting for the reset, after which it should be checked to see how long before the timeout, so you are sure that the watchdog bit after the correct interval. If counting the number of watchdog resets to decide whether the system should give up trying, then be sure that you do not inadvertently count the watchdog test reset as one of those.

16.9.3 Task timers

A task-timer strategy has four objectives in a multitasking or multithreading system: (1) to detect an operating system, (2) to detect an infinite loop in any of the tasks, (3) to detect deadlock involving two or more tasks, (4) to detect whether some lower-priority tasks are never running because higher-priority tasks are hogging the CPU.

Typically, not enough timing information is available on the possible paths of any given task to check for a minimum execution time, or to set the time limit on a task to be exactly the time taken for the longest path. Therefore, although all infinite loops are detected, an error that causes a loop to execute a number of extra iterations may go undetected by the designed task timer mechanism. A number of other considerations have to be taken into account to make any scheme feasible:

(a) the extra code added to the normal tasks (as distinct from a task created for monitoring tasks) must be small, to reduce the likelihood of becoming prone to errors itself;
(b) the amount of system resources used, especially CPU cycles, must be reasonable.

Most tasks have some minimum period during which they are required to run. A task may run in reaction to a timer that occurs at a regular interval. These tasks have a start point through which they pass in each execution loop. They are referred to as regular tasks. Other tasks respond to outside events, the frequency of which cannot be predicted, and are referred to as waiting tasks.

The watchdog can be used as a task timer. Its timeout can be chosen to be the maximum time during which all regular tasks have had a chance to run from their start point through one full loop back to their start point again. Each task has a flag that can have two values, ALIVE and UNKNOWN. The flag is later read and written by the monitor. The monitor's job is to wake up before the watchdog timeout expires and check the status of each flag. If all flags contain the value ALIVE, every task had its turn to execute and the watchdog may be kicked. Some tasks may have executed several loops and set their flag several times to ALIVE, which is acceptable. After kicking the watchdog, the monitor sets all of the flags to UNKNOWN. By the time the monitor task executes again, all of the UNKNOWN flags should have been overwritten with ALIVE.

16.9.4 Timer creation and expiration

An active timer will perform its synchronization action when its expiration time is reached. A timer can be created using the Timer-Create system call. It is bound to a task and to a clock device that will measure the passage of time for it. This is useful for ownership and proper tear-down of timer resources. If a task is terminated, then the timer resources owned by it are also terminated as well. It is also possible to explicitly terminate a timer using the Timer-Terminate system call.

Timers allow synchronization primarily through two interfaces. Timer-Sleep is a synchronous call, in which the caller specifies a time to wake up and indicates whether that time is relative or absolute. Relative times are less useful for real-time software since program-wide accuracy may be skewed by preemption. For example, a thread which samples data and then sleeps for 5 seconds may be preempted between sampling and sleeping, causing a steadily increasing skew from the correct time base. Timers that are terminated or canceled while sleeping return an error to the user. Another interface that can be named "Timer-Arm" provides an asynchronous interface to timers. Through it the user specifies an expiration time, an optional period, and a port which will receive expiration notification messages.

When the expiration time is reached, the kernel sends an asynchronous message containing the current time to this port, which then carries on stopping the related task that owns this timer. If the timer is specified as periodic, then it will rearm itself with the specified period relative to the last expiration time.

Last, the Timer-Cancel call allows the user to cancel the expiration of a pending timer. For periodic timers, only the current expiration, or all forthcoming expirations can be cancelled. These last facilities, periodic timers and partial cancelation, are important because they allow an efficient and correct implementation of periodic computation. This permits a user-level implementation of real-time periodic threads.

Problems

1. Please try to give a definition of real time operating systems.
2. In subsection 16.1.1, this textbook assumes that there are five basic requirements for real time operating systems. Based on your definition, please explain why these five items are necessary for an operating system to be a real time operating system.
3. Please plot two data flow diagrams for the task context switching implemented by real time operating systems with both single processor and multiprocessor platforms, respectively.
4. Imagine that you are working on designing a real time operating system written in the C++ programming language for a video game machine: please design all the Classes for this real time operating system, and plot a relation diagram between these Classes.
5. Please list the differences between standard and real time operating systems as far as possible.
6. Please explain the mechanisms for preemptive multitasking with single processor and multiprocessor platforms.
7. Is it necessary for any multitasking enabled operating system to have one task (process) as the "root task" which does not have a "parent" and is forever in the "running" state?
8. Please explain what hardware context switching for tasks is; and what software context switching for tasks is.
9. Can these scheduling algorithms, FCFS, RR, SJF and PS, be used for task scheduling for real time operating systems?
10. Please explain what an interrupt controller is and what an interrupt handler is. Which is hardware based, and which is software based?
11. Design a single interrupt stack and a multiple interrupt stack (just their layout).
12. Plot possible binary timing diagrams (with the HIGH signal as 1 and the LOW signal as 0) of the five registers of the interrupt controller given in Table 16.2 when a CPU handles a masked and a non masked interrupt.

13. The mechanism of maskable and non maskable interrupts is used to enable and disable hardware interrupts. How can a microprocessor enable and disable software interrupts? (hint: most operating systems set a software interrupt as an "exception event".)
14. If the interrupt stack expands into the interrupt vector, the target system will crash, unless some check is placed on the extension of the stack and some means exist to handle that error when it occurs. Please give an explanation of this problem.
15. What are the differences between the interrupt service routine and the interrupt service thread?
16. In Figure 16.15, suppose that the physical memory has 64 entries, and each entry has 64 bytes. Then, please extend these two virtual memories for both processes X and Y into 32 pages each of which has 64 bytes, respectively. Based on your designs for the mappings between these two virtual memories and the physical memory, please give the page table for process X, and the Page Table For process Y. (Hint: you probably need determine the data structure for the page table first.)
17. What are the differences between segmented and paged virtual memories?
18. Please give all the algorithms for dynamic allocation; explain their working mechanisms, and figure out which of them can be used for dynamic allocation in a RTOS.
19. The main purpose of memory protection is to prevent a process running on an operating system from accessing memory beyond that allocated to it. The main purpose of memory access control is to avoid concurrent access by more than one process (or task) to the same memory area. Why are both memory protection and memory access control necessary for a RTOS?
20. Based on Figure 16.17 and using standard C++, design an event notification service and define all its classes and methods (functions).
21. What data structure can be used for the "channel" message queue? What data structure can be used for the "pipe" message queue? Do both of them require static or dynamic memory allocation in run time?
22. Explain all the differences between synchronous and asynchronous message passing.
23. Three PCs (personal computer) connect to one printer. A file writing processes in Microsoft Word is running on each of these three PCs. Please write pseudocode in which semaphores are used to coordinate the processes' access to the printer.
24. It seems that both semaphore and timer are used by the task scheduler for task synchronization. However, they perform different functions in the task scheduler. Please identify which types of semaphore and timer can be used by the task scheduler. What are the semaphore functions in a task scheduler? What are the timer functions in a task scheduler?

Further Reading

S. Baskiyar. A survey of contemporary real time operating systems. Informatica 29 (2005), 233 240.

David Kalinsky, et al. Basic concepts of real time operating systems. http://www.linuxdevices.com/articles/. November 2007.

John Carbone. A SMP RTOS for the ARM MPCore Multiprocessor. ARM Developers Conference, 4(3). 2005.

Quadros Systems, Inc. (http://www.quadros.com). An Introduction to Real time Operating Systems. September 2006.

Wikipedia (http://en.wikipedia.org). Real time operating system. http://en.wikipedia.org/wiki/Real time operating system. Accessed: October 2008.

Arezou Mohammadi, Selim G. Akl. 2005. Technical Report No. 2005 499: Scheduling Algorithms for Real Time Systems. Queen's University of Canada.

Wikipedia (http://en.wikipedia.org). Computer Multitasking. http://en.wikipedia.org/wiki/Computer multitasking. Accessed: October 2008.

David A. Rusling. The Linux Kernel: http://www.linuxhq.com/guides/TLK/tlk toc.html. Accessed: October 2008.

Sun Microsystems (http://www.sun.com/). Interrupt handlers. http://docs.sun.com/app/docs/doc/801 6678/6i11oeldm?a=view. Accessed : November 2008.

Embedded Company (http://www.embedded.com). Introduction to interrupts. http://www.embedded.com/TechSearch/Search.jhtml?queryText=introduction+to +interrupts&site id=Embedded.com. Accessed: November 2008.

Wikipedia (http://en.wikipedia.org). Memory management. http://en.wikipedia.org/wiki/Memory management. Accessed: October 2008.

FreeBSD (http://www.freebsd.org). Memory management. http://www.freebsd.org/doc/en/books/design 44bsd/overview memory management.html. Accessed: October 2008.

CHAPTER

Distributed operating systems 17

Based on the control functions performed, a distributed control system can be architected into these hardware components: operator interfaces, I/O subsystem, connection buses, and field control units; and into these software modules: history modules, control modules, I/O modules, and network modules. These were discussed in detail in section 1.3 of this textbook.

In advanced industrial controls, the control units in distributed systems are digital, intelligent controllers or computers, containing microprocessors. From the computer point of view, both distributed control systems and distributed computer systems have similar hardware and software architectures. Therefore, distributed operating systems are required by both distributed control and computer systems. The term distributed system is therefore used here for both distributed control systems and distributed computer systems.

A distributed system consists of a set of computers that communicate with each other using hardware and software interconnecting devices. Generally, distributed systems exist in two types of hardware architectures. The first type is multiprocessor architecture, in which two or more microprocessors or CPUs are fully connected with buses or switches and share a common memory. In this arrangement, every microprocessor or CPU has equal access to the entire physical memory, and communication among them uses the shared memory model. It is found in some supercomputers such as the Cray-T3E, and in some equipment such as multifunction copiers.

The second type is multicomputer architecture, in which several independent computers are physically connected with hardware, and dynamically coupled with software to make up a computer network. LANs and computer clusters within a corporation. building or office are typical networks where multicomputer architectures are used.

All the software for distributed systems requires a kernel that plays the same role as that played by the operating system in single-processor or centralized systems. This is defined as a distributed operating system, and it manages the shared resources, schedules the processes and coordinates communications between the elements of the distributed system. In fact, distributed operating systems are just an extension of the distributed system architecture of multitasking operating systems applied to centralized system architectures.

There are two categories of such systems. The first is the multiprocessor operating system, often just a regular operating system. Nevertheless, they some have unique features including process synchronization, resource management, and scheduling.

The second kind is the multicomputer operating system, which may be considered to be either a computer network (loosely coupled computers) or a computer cluster (tightly coupled computers). This textbook explains three types of multicomputer operating systems: cluster operating systems, network operating systems, and parallel operating systems.

There are four mechanisms behind distributed operating systems: (1) distributed and parallel process management; (2) distributed and parallel file systems; (3) distributed and shared memory allocations; (4) interior, local and remote communications see section 17.3 for further details.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10017-8

17.1 MULTIPROCESSOR OPERATING SYSTEMS

17.1.1 Multiprocessor hardware and software models

One main property of a multiprocessor is its interconnection network, which links the processors, memory modules and the other devices in a system. An interconnection network, which may be static or dynamic, facilitates communication among processors and memory modules. A few sample interconnection networks are: time-shared or common buses, crossbar switches, hierarchical switches, and multistage networks.Although in all multiprocessors, every CPU can address all memory words, in some, every memory word can be read as equally quickly. Depending on the coupling of processors and memory, multiprocessors may be broadly divided into two major categories: shared-memory, and non-remote memory access (NORMA) multiprocessors.

(1) Shared-memory multiprocessors

In a shared-memory multiprocessor, all main memory is accessible to and shared by all processors, as shown in Figure 17.1. On the basis of the cost of accessing shared memory, shared memory multiprocessors are classified as:

(a) Uniform memory access (UMA) multiprocessors. In the UMA architecture, the access time to shared memory is the same for all processors. The simplest have bus-based architecture, as illustrated in Figure 17.2. Figure 17.2(a) is a single-bus architecture in which two or more CPUs and one or more memory modules all use the same bus for communication. In Figure 17.2(b), each CPU has an additional cache, which can be inside the CPU chip, next to it, on the processor board, or some combination of all three. Figure 17.2(c) shows a bus with caching and memories, in which each CPU has not only a cache, but also a local, private memory which it accesses over a dedicated (private) bus. In addition to the bus-based

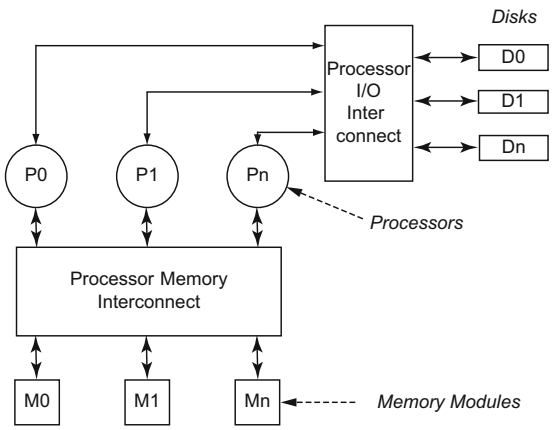


FIGURE 17.1

The shared memory architecture for multiprocessors.

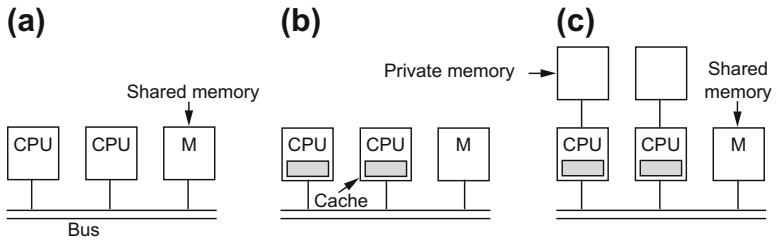


FIGURE 17.2

Three bus-based multiprocessors: (a) without caching; (b) with caching; (c) with caching and private memories.

architectures, there are two other designs, which use crossbar switches, or multistage switching networks. However, these two designs may not require different mechanisms and semantics from the bus-based architectures in their operating systems.

(b) Non-uniform memory access (NUMA) multiprocessors. In the NUMA architecture, all physical memory in the system is partitioned into modules, each of which is local to and associated with a specific processor. As a result, access time for local memory is less than that for nonlocal memory. Nearly all CPU architectures use a small amount of very fast non-shared memory, known as cache, to exploit locality of reference in memory accesses. With NUMA, maintaining cache coherence across shared memory has a significant overhead. One can view NUMA as a very tightly coupled form of cluster computing. The addition of virtual memory paging to cluster architecture can allow the implementation of NUMA entirely in software where no NUMA hardware exists. However, the inter-node latency of software-based NUMA remains several orders of magnitude greater than that of hardware-based NUMA.

UMA architectures are the most common parallel machines, in part because most such machines are simply used as high-throughput multiple-programmed, multiple-user timesharing machines, rather than as execution vehicles for single, large-scale parallel programs. Interestingly, although all memory is accessed via a single shared bus, even UMA machines often have NUMA characteristics, because individual processors access shared memory via local caches. Cache misses and cache flushing can result in effectively non-uniform memory access times. Furthermore, bus contention may aggravate variability in memory access times, and scalability is limited, in that the shared global bus imposes limits on the maximum number of processors and memory modules it can accommodate.

NUMA architecture addresses the scalability problem by attaching local memory to each processor. Processors directly access local memory and communicate with each other and with remote memory modules through an interconnection switch. One type of switch is an interconnection network consisting of multiple levels of internal nodes, where systems are scaled by addition of internal switch nodes. A second type of switch consists of a hierarchical set of buses, where access times to remote memory depend on either the number of internal switch nodes on the access path between the processor and the memory or on the number of traversed system buses. Because NUMA architecture allows a large number of processors in a single machine, many experimental, large-scale multiprocessors are NUMA machines.

(2) Non-remote memory access (NORMA) multiprocessors

In this class of architectures, each processor has its own local memory that is not shared by other processors in the system. Some modern Intel multiprocessors and HP workstation clusters are examples of non-shared memory multiprocessors. Workstation clusters differ from hypercube or mesh machines, in that the latter typically offer specialized hardware for low-latency inter-machine communication and also for implementation of selected global operations such as global synchronization, addition, or broadcast.

NORMA multiprocessors are the simplest to design and build, and have become the architecture of choice for current supercomputers such as the Intel Paragon, the Cray series of high-performance machines, and others. In the simplest case, a collection of workstations on a local area network constitutes a NORMA multiprocessor. A typical NORMA multiprocessor consists of a number of processors interconnected on a high-speed bus or network cables; the topology of interconnection varies between applications.

One major difference between NORMA multiprocessors and NUMA is that there is no hardware support for direct access to remote memory modules. As a result, the formes are more loosely coupled. However, recent advances are leading to trade-offs in remote to local memory access times for NORMA machines (e.g., roughly 1:500 for local vs. remote memory access times) that can approximate those achieved for shared-memory machines (roughly 1:100). This suggests that future NUMA or NORMA parallel machines will require similar operating system and programming tool support in order to achieve high-performance parallelism.

The variety of different kinds of multiprocessor architectures, coupled with diverse application requirements, have resulted in many different designs, goals, features, and implementations of multiprocessor operating systems, both in university research projects and in the commercial domain. To introduce these, it is necessary to understand three important concepts: processor symmetry, processor instruction, and processor data-stream.

In a multiprocessing system, all CPUs may be equal, or some may be reserved for special purposes. A combination of multiprocessor hardware and operating system software design determines the symmetry (or lack thereof) in a given system. For example, hardware or software considerations may require that only one CPU responds to all hardware interrupts, whereas all other work in the system may be distributed equally among the remainder, or execution of kernel-mode code may be restricted to only one processor (either a specific processor, or only one processor at a time), whereas user-mode code may be executed in any combination of processors. Multiprocessing systems are often easier to design if such restrictions are imposed, but they tend to be less efficient than systems in which all CPUs are utilized equally.

Systems that treat all CPUs equally are called symmetric multiprocessing (SMP) systems. In systems where all CPUs are not equal, resources may be divided in a number of ways, including asymmetric multiprocessing (ASMP), non-uniform memory access (NUMA) multiprocessing, and clustered multiprocessing.

In multiprocessing, the processors can be used to execute a single sequence of instructions in multiple contexts (single instruction, multiple data or SIMD, often used in vector processing); or multiple sequences of instructions in a single context (multiple instruction, single data or MISD, used for redundancy in fail-safe systems and sometimes applied to describe pipelined processors or hyperthreading); or multiple sequences of instructions in multiple contexts (multiple instruction, multiple data or MIMD).

(a) SIMD multiprocessing is well suited to parallel or vector processing, as mentioned, since a very large set of data can be divided into parts that are individually subject to identical but independent operations. A single instruction stream directs the operation of multiple processing units to perform the same manipulations simultaneously on potentially large amounts of data. Figure 17.3(a) provides an illustration of SIMD multiprocessing.

(b) MISD is a type of parallel computing architecture where many functional units perform different operations on the same data. Pipeline architectures belong to this type, though a purist might say that the data are different after processing by each stage in the pipeline. Fault-tolerant computers executing the same instructions redundantly in order to detect and mask errors, in a manner known as task replication, may also be considered to belong to this type. Not many instances of this architecture exist, as MIMD and SIMD are often more appropriate for common data parallel techniques. Specifically, they allow better scaling and use of computational resources than MISD does. Figure 17.3(b) provides an illustration of MISD multiprocessing.

(c) MIMD is a technique employed to achieve parallelism. Machines using MIMD have a number of processors that function asynchronously and independently. At any time, different processors may be executing different instructions on different pieces of data. Such machine may be used in a number of application areas, such as computer-aided design and computer-aided manufacturing, simulation, modeling, and as communication switches. MIMD machines can be of either shared-memory or distributed-memory categories based on how their processors access memory. Shared-memory machines may be of the bus-based, extended, or hierarchical type, while distributed-memory machines may have hypercube or mesh interconnection schemes. Figure 17.3(c) provides an illustration of MIMD multiprocessing.

17.1.2 Processor scheduling

The basic functionality of a multiprocessor operating system must include most of what is present in a multitasking operating systems for single-processor machines. However, complexities arise due to

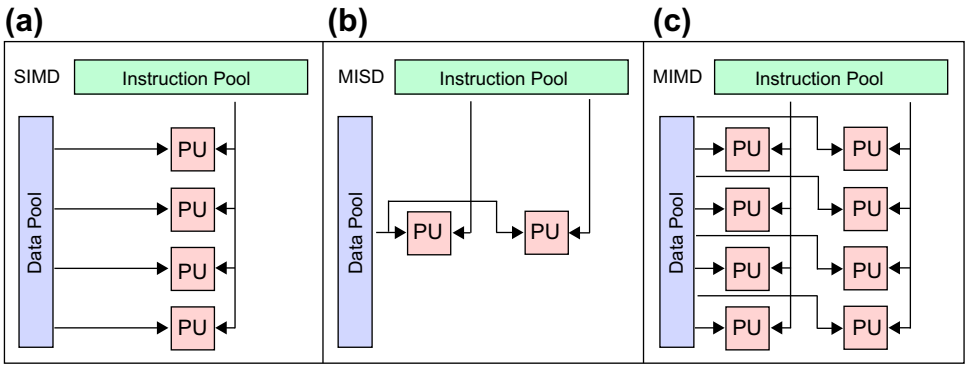


FIGURE 17.3

The software implementations of multiprocessors: (a) SIMD (single instruction, multiple data); (b) MISD (multiple instruction, single data); (c) MIMD (multiple instruction, multiple data).

the additional capabilities of multiprocessor hardware and, more importantly, due to the extreme requirements of performance imposed on the multiprocessor operating system. The classic functions of an operating system include the creation and management of dynamic entities such as jobs, processes and tasks (or threads, as they may be termed hereafter). The effectiveness of multiprocessor computing depends on the performance of the primitives used to manage processors and to share resources.

(1) Heavyweight processes to lightweight processes

One way to express parallelism is by using UNIX-like processes sharing parts of their address spaces. Such a process consists of a single address space and a single task of control. Those operating systems' kernels which support such processes do not distinguish between a task and its address space; they are sometimes referred to as heavyweight tasks. The parallelism expressed using heavyweight tasks is coarse-grained and is too inefficient for general-purpose parallel programming because of the following reasons:

(a) Since the operating system kernel treats a task and its address space as a single entity, they are created, scheduled, and destroyed together. As a result, the creation and deletion of heavyweight tasks is expensive.

(b) Reallocating a processor to a different address space (this operation is a context-switch) is expensive. There is an initial scheduling cost to decide the address space to which the processor should be reallocated, and next, there is a cost for updating the virtual memory mapping registers and transferring the processor between address spaces. Finally, there is a long-term cost associated with cache due to the address space change.

In many operating system kernels, therefore, address spaces and tasks are decoupled, so that a single address space can have more than one execution task. Such tasks are referred to as middleweight or kernel-level tasks when they are managed by the operating system kernel. The advantages of middleweight tasks are: (1) the kernel can directly schedule an application's task on the available physical processors; (2) these kernel-level tasks offer a general programming interface to the application.

However, kernel-level tasks also exhibit some problems that can make them impractical for use in fine-grained parallel programs. The first of these is that the cost of generality of kernel-level tasks is not acceptable for fine-grained parallel applications. For example, saving and restoring the floating point context in a context switch are expensive and may be unnecessary for a specific application program. The second problem is that a relatively costly protected kernel call is required to invoke any task management operation, including task synchronization, and thirdly a single model represented by one style of kernel-level task is unlikely to have an implementation that is efficient for all parallel programs.

To address the above, some operating system designers have turned to user-level tasks, also known as lightweight tasks. These are managed by run-time library routines linked into each application. A management operation on a user-level task does not require an expensive kernel call, and such tasks enable an application program to use a task management system most appropriate to the problem domain under development.

A lightweight task generally executes in the context of a middleweight or a heavyweight task. Specifically, the task library schedules lightweight tasks on top of middleweight or heavyweight ones,

which in turn are scheduled by the kernel on the available physical processors. Such a two-level scheduling policy has some inherent problems:

(a) User-level tasks, typically, do not have any knowledge of kernel events (e.g., processor preemption, I/O blocking and resuming, etc.). As a result, the application library cannot schedule a task on a "just idle" processor.

(b) When the number of runnable kernel-level tasks in a single address space is greater than the number of available processors, these tasks must be multiplexed. This implies that user-level tasks built on top of kernel-level tasks are actually scheduled by the kernel's task scheduler, which nevertheless has little or no knowledge of the application's scheduling requirements or current state.

Problems with multi-level scheduling arise from the lack of information flow between the different levels. For two-level scheduling, they can be solved by explicit vectoring of kernel events to the user-level task scheduler, using up-calls called scheduler activations, and by notifying the kernel of user-level events affecting processor allocation. Another solution dynamically controls the number of processes used by applications, which will be discussed later on.

Similarly, a set of kernel mechanisms have been proposed to implement "first-class user-level" tasks addressing the above problem. These mechanisms include shared kernel and user data structures (for asynchronous communication between the kernel and the user), software interrupts (for events that might require action on the part of a user-level scheduler), and a scheduler interface convention that facilitates interactions in user space between dissimilar kinds of tasks. There is another solution that also explores data structure alternatives when implementing user-level task packages. Here, alternative implementations are evaluated in performance for task run queues, idle processor queues, and for spinlock management.

We are not aware of general solutions to the multi-level scheduling problem, other than the actual exchange or configuration of the operating system's tasks scheduler by application programs, as is often done in real-time systems.

(2) Scheduling policies

A scheduling policy is required by a multiprocessor operating system to allocate available time and processors to a job or a process, either statically or dynamically. Processor load balancing is considered to be a part of a scheduling policy.

These are operated by multiprocessor schedulers. As with other operating system services for parallel machines, schedulers themselves must be structured to be scalable to different-size target machines and to different application requirements.

There are two scheduling methods: static and dynamic. A static scheduler makes a one-time decision per job of how many processors to allocate. Once decided, the job is guaranteed to have exactly that number of processors whenever it is active. This approach offers low run-time scheduling overhead, but it also assumes a stable parallel application. This is a reasonable assumption for many large-scale scientific applications in which parallelism is derived by decomposition of regular data domains.

Recent work, however, is focusing more on dynamic scheduling because most complex large-scale parallel applications exhibit irregular data domains or changes in domain decompositions over time, so that a static processor allocation rapidly becomes inefficient; and also because large-scale

parallel machines are often used in multi-user mode, so that scheduling must take into account the requirements of multiple parallel applications sharing a single machine. The dynamic policy occasionally exhibits a performance penalty when overhead values are very large. One reason for such performance degradation is a possible high rate of processor reallocation. Hence, some solutions have been suggested for dampening the rate of processor allocation and release, thereby reducing the rate of "useless processor exchange". However, such a modification to the dynamic policy was found to be detrimental to performance. As for single-processor schedulers, multiprocessor schedulers can be classified as preemptive or non-preemptive. A scheduler can also be classified according to its scheduling granularity, which is determined by the executable unit being scheduled (for example, schedulers differ in that they may schedule individual or groups of processes).

In this subsection, we focus on dynamic scheduling, and on scheduling for shared memory machines, where variations in distances between different processors on the parallel machine are not considered. A few well-accepted multiprocessor scheduling policies are now reviewed.

(a) Single shared ready queue

Research addressing UMA multiprocessors has typically assumed the use of a single ready queue shared by all processors. With this queue, scheduling policies such as first-come first-served (FCFS) or shortest job first (SJF) are easily implemented, and have been evaluated in the literature. More interesting to us are schedulers and scheduling policies directly addressing the primary requirement of a parallel program: if performance improvements are to be attained by use of parallelism, then the program's processes must be scheduled to execute in parallel.

(b) Co-scheduling

The goal of co-scheduling (or gang scheduling) is to achieve a high degree of simultaneous execution of processes belonging to a single job. This is particularly useful for a parallel application with cooperating processes that communicate frequently. A co-scheduling policy schedules the runnable processes of a job to run simultaneously on different processors. Job preemption implies the simultaneous preemption of all of its processes. Effectively, the system context-switches between jobs. The scheduling algorithms used in the co-scheduling policy will be explained later in this subsection.

(c) Round-robin (RR) scheduling

Two versions of RR scheduling exist for multiprocessors. The first is a straightforward extension of the single-processor round-robin scheduling policy, which appends its processes to the end of the shared process queue. A round-robin scheduling policy is then invoked on the process queue. The second version uses jobs rather than processes as the scheduling unit, so the shared process queue is replaced by a shared job queue. Each entry of this queue itself contains a queue holding its processes. The scheduling algorithms supporting the RR policy are discussed in section (3) below.

(d) Dynamic partitioning

Dynamic partitioning (also known as the process control with processor partitioning) policy has a goal of minimizing context switches, so that less time is spent rebuilding a processor's cache. This approach is based on the hypothesis that an application performs best when the number of runnable processes is the same as the number of processors. As a result, each job is dynamically allocated an equal fraction of the total number of processors, but none is allocated more processors than it has runnable processes.

Each application program periodically polls a scheduling server to determine the number of processes it should ideally run. If the ideal number is less than the actual number, the process suspends some of its processes, if possible. If the ideal number is greater than the actual number, a process wakes up a previously suspended process. This policy has limited generality since it requires interactions between user processes and the operating system scheduler, and it also requires user programs to be written such that their processes can be suspended and woken up during execution.

(e) Hand-off scheduling

A kernel level scheduler accepts user hints. Two kinds of hints exist: (1) discouragement hints, which is used to discourage the scheduler from running the current task; it may be either mild and strong, or weak; (2) hand-off hints, used to suggest that the scheduler runs a specific task. When using a hand-off hint, the current task hands off the processor to another task without intermediate scheduler interference. Such schedulers are better known as hand-off schedulers. Experiments with scheduling hints have shown that they can be used to improve program performance, particularly when program synchronization is exploited (e.g., the requester task hands off the processor to the holder of the lock) and when interprocess communication takes place (e.g., the sender hands the processor off to the receiver).

(3) Scheduling algorithms

There are three different co-scheduling algorithms: matrix, continuous and undivided. In a matrix algorithm, processes of arriving jobs are arranged in a matrix with P columns and a certain number of rows, where P is the total number of processors in the system. This arrangement is such that all the processes in a job reside in a same row. The scheduling algorithm uses a round-robin mechanism to multiplex the system between different rows of the matrix, so that all the processes in a row are co-scheduled.

A problem with the matrix algorithm is that a hole in the matrix may result in a processor being idle even though there are runnable processes. The continuous algorithm addresses this problem by arranging all processes in a linear sequence of activity slots. The algorithm considers a window of P consecutive positions in the sequence at a particular moment. When a new job arrives, the window is checked to see whether there are enough empty slots to satisfy its requirements. If not, the window is moved one or more positions to the right, until the leftmost activity slot in the window is empty but the slot just outside the window to the left is full. This process is repeated until a suitable window position is found to contain the entire job, or the end of the linear sequence is reached. Scheduling consists of moving the window to the right at the beginning of each time slice until the leftmost process in the window is the leftmost process of a job that was not co-scheduled in the previous time slice.

The most serious problem with the continuous algorithm is analogous to external fragmentation in a segmentation system. A new job may be split into fragments, which can result in unfair scheduling for large, split jobs compared to small contiguous jobs. This issue has been addressed by designing an undivided algorithm, which is identical to the continuous algorithm except that all of the processes of each new job are required to be contiguous in the linear activity sequence. This algorithm can be slightly modified to eliminate some of its performance problems: when a job arrives, its processes are appended to the end of a linked list of processes. In this case, scheduling is done by moving a window of length equal to the number of processors over the linked list. Each process in the window receives one quantum of service on a processor. At the end of this, the window is moved down the linked list until its first slot is over the first process of a job that was not completely co-scheduled in the previous

quantum. When a process within the window is not runnable, the window is extended by one process and the non-runnable process is not scheduled. All processors that switch processes at the end of a quantum do so at the same time. A second algorithm modification improves expected performance for correlated workloads. This modification applies to the movement of the window. At the end of each of quantum, the window is only moved to the first process of the next job, even if it was co-scheduled in the previous time slice.

Scheduling policy with a round robin is done on the jobs. The job in the front of the queue receives P quanta of size Q, where P is the number of processors in the system and Q is the quantum size. If a job has fewer processes than P, then the total quanta size, which is equal to PQ, is divided equally among the processes. If the number of processes in a job exceeds P, then there are two choices. The first choice is same as the previous case, i.e., divide the total quanta size PQ equally among all processes. Alternatively, one can choose P processes from the job in a round-robin fashion, each process executing for one quantum. The first alternative has more scheduling overhead than the second.

(4) Some remarks

The performance of an application worsens considerably when the number of processes exceeds the total number of processors. This decreased performance may come from several factors:

(a) A process may be preempted while inside a spinlock-controlled critical section, while the other processes of the same application "busy wait" to enter the critical section. This problem is particularly acute for fine-grain parallel programs. Identical problems arise when programs' processes are engaged in producer/consumer relationships.

(b) Frequent context switches occur when the number of processes greatly exceeds the number of processors.

(c) When a processor is interleaved between multiple address spaces, cache misses can be a major source of performance degradation. Careful application design and co-scheduling may handle problems associated with spinlock-controlled critical sections, and those with producer-consumer processes, but they do not address performance degradation due to cache corruption or frequent context switches.

A more direct solution is proposed, which describes a task scheduler that avoids preempting processes inside critical sections. This approach combines co-scheduling and preemption avoidance for critical sections and combines multiple processes to form a group. The scheduling policy of a group can be set so that either all processes in the group are scheduled and preempted simultaneously, or individual processes are scheduled and preempted normally, or processes in the group are never preempted. An individual process may choose to override its group scheduling policy, which is flexible, but leaves specific solutions of the critical section problem to user code. Problems with cache corruption and context-switch frequency are addressed by evaluating the performance of several multiprocessor scheduling policies based on the notion of processor affinity. A process's processor affinity is based on the contents of the processor's cache. The basic policy schedules a process on a processor on which it last executed, hoping that a large percentage of its working set is still present in the processor's cache. Since the policy inherently discourages process migration, it may lead to severe load imbalance. Similarly, affinity (for local memory) also plays a vital role in scheduling processes in a NUMA machine; the context of a process resides mostly near the processor on which the process executed last. The effect of cache affinity on kernel processor scheduling discipline for multiple-programmed,

shared-memory multiprocessors shows that the cache effects due to processor reallocation can be significant.

17.1.3 Memory management

Memory management for UMA multiprocessors is conceptually similar to that in a multitasking operating system for a single-processor machine. As mentioned earlier, in UMA architecture, memory access times are equal for all processors, but the underlying architecture typically supports some degree of parallelism in global memory access. As a result, even for UMA machines, operating system writers must exploit the available hardware parallelism when implementing efficient memory management. More interesting problems arise for NUMA and NORMA machines.

(1) Shared virtual memory

Memory management services and implementations are generally dependent on the operating system, as well as the underlying machine architecture. Some effort has focused on designing memory management functionalities and interfaces which are independent of the machine architecture and the operating system kernel. For example, virtual memory management is machine- and operating-system-independent. In the operating systems that follow this design, the machine-dependent portion of the virtual memory subsystem is implemented as a separate module. All information necessary for the management of virtual memory is maintained in machine-independent data structures that contain only the mappings necessary to run the current mix of programs. Similarly, a scalable, kernel-independent, generic memory management interface is suitable for various architectures (e.g. paged and or segmented) and implementation schemes.

Some operating systems allow applications to specify the protection level (inaccessible, read-only, read-write) of pages, and allow user programs to handle protection violations. Several user-level algorithms have been developed that make use of page-protection techniques, and analyze their common characteristics, in an attempt to identify the virtual memory primitives the operating system should provide to user processes.

(2) NUMA and NORMA memory management

A NUMA multiprocessor organization leads to memory management design choices that differ markedly from those that are common in systems designed for single processors and UMA multiprocessors. NUMA architectures implementing a shared-memory programming model typically expose the existing memory access hierarchy to the application program.

We will briefly discuss several memory management algorithms for NUMA multiprocessors. Those described below are categorized by whether they migrate and/or replicate data. An algorithm would migrate data to the site where they are accessed in an attempt to exploit locality in data accesses, and decrease the number of remote accesses. The others replicate data so that multiple read accesses can happen at the same time, using local accesses.

(a) Migration algorithm

In the migration algorithm, the data are always migrated to the local memory of the processor which accesses them. If an application exhibits a high locality of reference, the cost of data migration is

amortized over multiple accesses. Such an algorithm may cause thrashing of pages between local memories. One disadvantage of the migration algorithm is that only the tasks on one processor can access the data efficiently. An access from a second processor may cause another migration.

(b) Read-replication algorithm

Replication reduces the average cost of read operations, since it allows a read to be simultaneously executed locally at multiple processors. However, a few write operations become more expensive, since a replica may have to be invalidated or updated to maintain consistency. If the ratio of reads over writes is large, the extra expense of the write operation may be offset. Replication can be naturally added to the migration algorithm for better performance.

(c) Full-replication algorithm

Full replication allows data blocks to be replicated even while being written to, so keeping the data copies consistent is a major concern. A number of algorithms are available for this purpose. One of them is similar to the write-update algorithm for cache consistency. Some specific NUMA memory management schemes are described for individual parallel operating systems in section 17.2.

(d) Page placement

A page placement mechanism is implemented to automatically assign pages of virtual memory to appropriately located physical memory in the operating system on multiprocessor machines. A simple strategy uses local memory as a cache over global, managing consistency with a directory-based ownership protocol similar to that used for distributed shared virtual memory. Dynamic page replacement policies for NUMA multiprocessors support both migration and replication, use a directory-based invalidation scheme to ensure the coherence of replicated pages, and use a freeze and defrost strategy to control page bouncing. Such a parameterized NUMA memory management policy can be tuned for architectural as well as application differences.

(e) Weak memory

Some research has explored how shared memory may be represented on multiprocessor machines such that its performance can approximate that of message-passing systems. For example, some memory models exploit the fact that synchronization is used to control access to a shared state, which allows the underlying system to weaken memory consistency requirements. The resulting, weakened shared memory abstraction presented to programmers may be implemented efficiently, because strong consistency, and therefore, inter-processor communication is not required for all memory accesses. Other models of shared memory that were developed for distributed architectures exploit programmer directives to reduce the cost of coherence maintenance, or they provide explicit primitives with which users can maintain application-specific notions of coherence of shared state.

17.1.4 Process control

When multiple cooperating processes execute simultaneously within a multiprocessor, synchronization primitives are needed for concurrency control. When multiple processes share an address space, synchronization is required for shared memory consistency. Let us now take a close look at how this synchronization actually works in a multiprocessor. To start with, if a process on a single processor

makes a system call that requires accessing some critical kernel table, the kernel code can simply disable interrupts before touching the table. It can then do its work knowing that it will be able to finish without any other process sneaking in and touching the table before it is finished. On a multiprocessor, disabling interrupts affects only the CPU doing the disable. Other CPUs continue to run and can still touch the critical table. As a consequence, a proper protocol must be used and respected by all CPUs to guarantee that mutual exclusion works.

Two fundamental semantics for single-processor operating systems also work for multiprocessor synchronization: mutual exclusion and event broker, discussed in detail in Chapter 16. This subsection briefly reviews some common and efficient synchronization constructs supported by recent multiprocessor operating systems, where the purpose of mutual exclusion is that when one process in one processor locks a critical section, all other processes, running anywhere, also have access blocked to this critical section.

(1) Locks

Multiprocessor operating systems typically support multiple types of lock.

(a) Spin and blocking locks

Spin locks are the most primitive type, and can be based on different mechanisms including queue locking, ticket lock and priority locking. When a lock is busy, a waiting process spins (busy-waits) until it is released. Most hardware supports spin locks by specific instructions in their instruction sets; each of these instructions allows one indivisible blocking operation (atomic operation). When using a blocking lock, a waiting process (also called a contender process) blocks until awakened by the process releasing the lock. Such locks are also known as mutex locks.

(b) Read-write locks

A read-write lock maintains a pair of associated locks, one for read-only operations and one for write-only. The read lock may be held simultaneously by multiple reader threads, as long as there are no writers. The write lock is exclusive. A read-write lock allows either multiple readers or a single writer to enter a critical section at the same time. The waiting processes may either spin or block depending on whether the lock is implemented as a spinning read-write lock or a blocking read-write lock.

(c) Configurable locks

These locks allow applications to alter the waiting (spin, block or both) mechanism dynamically and as well as the request-handling mechanism (how the lock is scheduled). Configurable locks demonstrate that combined locks (locks that both spin and block while waiting) improve application performance considerably, compared to simple spin or blocking locks. Furthermore, hints from lock owners may be used to configure a lock, and improve its waiting strategy (advisory or speculative locks).

(d) Barrier locks

A barrier lock implements a barrier in a parallel program, which provide a means of ensuring that no processes advance beyond a particular point until all have arrived there. Once a process reaches a barrier, it is allowed to proceed if and only if all other cooperating processes reach the barrier. With

a barrier lock, a waiting process may either spin or block depending on the implementation of the lock. There are three main barrier constructs: centralized barrier, three barrier, and dissemination barrier.

(e) Structured locks

In distributed-memory machines, multiprocessor operating system constructs (e.g., I/O, exception handling, multicast communications, etc.) are physically distributed in order to offer efficient access to the global operating system functionalities. Synchronization is no exception because it is a computation that must be performed globally for many physically distributed processes and processors. As a result, this must be performed using explicit communication structures such as rings or spanning trees, which touch upon all members of the group of processes to be synchronized. In essence, a lock in a distributed-memory machine is a fragmented and distributed abstraction shared among several independently executable processes, where its importance is demonstrated by explicit support in hardware in several parallel machines. However, in contrast to the operating systems for UMA and NUMA multiprocessors, synchronization abstractions for distributed-memory machines can often be optimized substantially if they can be made programmable, or if it can be combined with other communications being performed in application programs.

(2) Other synchronization constructs

There are three further synchronization constructs, as explained below, used for multiprocessor operating systems.

(a) Test-and-set instructions

The test-and-set instruction is used to both test and (conditionally) write to a memory location as part of a single indivisible (atomic) operation. This means setting a value, but first performing some test (such as, is the value equal to another given value). If the test fails, the value is not set. If multiple processes may access the same memory and if a process is currently performing a test-and-set, no other process may begin another test-and-set until the first process is done. CPUs may use test-and-set instructions offered by other electronic components, such as dual-port RAM (DPRAM); CPUs may also offer such an instruction themselves.

(b) Condition variables

Condition variables make it possible for a task (or thread) to suspend its execution while awaiting an action by some other task. A condition variable is associated with some shared variables protected by a mutex and a predicate (based on the shared variables). A process acquires the mutex and evaluates the predicate. If it is not satisfied, the process waits on the condition variable. Such a wait atomically releases the mutex and suspends execution of the process. After a process has changed the shared variables so that the predicate is satisfied, it may signal a waiting task. This allows blocked tasks to resume action, to re-acquire the mutex, and to re-evaluate the predicate to determine whether to proceed or wait.

(c) Event brokers

Events are mainly used to control task orderings. A process may wait on an event; it blocks until the event occurs. Upon event occurrence, a signal wakes up one or all waiting processes. Events come in different flavors. A state (happened or not happened) may or may not be associated with an event. An

account may be associated with an event, which enables a process to wait for a particular occurrence of an event. More complicated event structures have been shown to be useful for several application domains and target machines, most prominently including the event-handling facilities for active messages or the synchronization points designed for real-time applications.

(3) Interprocess communications

Cooperating processes or tasks in a multiprocessor environment often communicate to help with synchronization and concurrency. Such communication employs one of two schemes: shared variables or message passing. As mentioned earlier, when two processes communicate using shared memory, synchronization is required to guarantee memory consistency. The previous section described a few popular synchronization primitives, whilst this one focuses on interprocess communication without using explicit shared variables.

In a shared memory multiprocessor, message-passing primitives between disjoint address spaces may be implemented using global memory. Exchange of messages is a more abstract form of communication than accessing shared memory locations. Message passing subsumes communication, buffering, and synchronization. Multiprocessor operating systems have experimented with a large variety of different communication abstractions, including ports, mailboxes, links, etc.

From an implementation point of view, such abstractions are kernel-handled message buffers, which may be either unidirectional or bidirectional. A process may send to or may receive messages from them. There may be rights (such as send, receive, or ownership rights) associated with these entities. Different operating systems define different semantics on these abstractions.

The two basic communication primitives in all such abstractions are both send and receive, which can again come in many different flavors. Sends and receives may be blocking (a process invoking a primitive blocks until the operation is complete), or they may be non-blocking (a process does not wait for the communication to be complete), or they may be conditional vs. unconditional. Communication between processes using these primitives may be synchronous or asynchronous, etc.

Many issues must be considered when designing an interprocess communication mechanism; they are reviewed in numerous surveys of distributed operating systems, and are not discussed in detail here. Issues include (1) whether the underlying hardware supports reliable or unreliable communication, (2) whether send and receives are blocking or non-blocking, (3) whether messages are typed or un-typed and of variable or fixed length, (4) how message queues can be kept short, (5) how to handle queue overflows, (6) how to support message priority (it may be necessary that some messages are handled at higher priorities than others), (7) how to transmit names, (8) protection issues, and (9) how kernel and user programs must interact to result in efficient message transfers. Communication issues specific to hypercube or mesh machines are reviewed elsewhere.

Most operating systems for shared-memory multiprocessor support cross-address space remote procedure calls (RPC) as a means of interprocess communication. RPC is a higher-level abstraction than message passing, that hides the message communication layer beneath a procedure call layer. It allows efficient and secure communications. Furthermore, cross-address space RPC can be made to look identical to cross-machine RPC, except that messages do not go out over the network and, in most cases, only one operating system kernel is involved in RPC processing. Otherwise, the same basic paradigm for control and data transfer is used. Messages are sent by way of the kernel between independent tasks bound to different address spaces. The use and performance of cross-address space RPC is discussed extensively in section 17.2.

17.2 MULTICOMPUTER OPERATING SYSTEMS

A multicomputer system may be either a network (loosely coupled computers) or a cluster (tightly coupled computers).

A computer network is an open system in which two or more computers are connected together to share resources such as hardware, data, and software. Most common are the local area network (LAN) and the wide area network (WAN). A LAN can range from a few computers in a small office to several thousand computers spread throughout dozens of buildings on a university campus or in an industrial park. Expand this latter scenario to encompass multiple geographic locations, possibly on different continents, and you have a WAN.

A computer cluster is a group of physically linked computers which work closely so that in many respects they behave as a single computer. A cluster consists of multiple stand-alone computers acting in parallel across a local high-speed network; thus clustered computers are usually much more tightly coupled. Its components are commonly, but not always, connected to each other through fast local area networks. An example of computer clusters is Columbia in NASA in the United States, which is a new supercomputer, built of 20 SGI Altix clusters, giving a total of 10,240 CPUs (Figure 17.4).

In Chapter 10, we discussed some industrial control networks including CAN, SCADA, Ethernet and LAN. CAN and Ethernet networks are tightly coupled networks similar to computer clusters, while SCADA and LAN networks are more loosely coupled, similar to computer networks.

FIGURE 17.4

Columbia is the new supercomputer, built of 20 SGI Altix clusters, a total of 10,240 CPUs. Source: http://www.nas.nasa.gov/News/Images/columbia_3.html.

This section will explain the operating systems used in these multicomputer systems, including cluster, network and parallel operating systems.

17.2.1 Cluster operating systems

In a distributed operating system, it is imperative to have a single system image so that users can interact with the system as if it were a single computer. It is also necessary to implement fault tolerance or error recovery features; that is, when a node fails, or is removed from the cluster, the rest of the processes running on the cluster can continue. The cluster operating system should be scalable, and should make the system highly available.

The single system image should implement a single entry point to the cluster, a single file hierarchy, a single control point, virtual networking, a single memory space, a single job management system, and a single user interface. Furthermore, for the high availability of the cluster, the operating system should feature a single I/O space, a single process space, and process migration.

Over the last three decades, researchers have constructed many prototype and production-quality cluster operating systems. We will now outline two of these cluster operating systems: Solaris MC and MOSIX.

(1) Solaris MC

Solaris MC is a prototype, distributed operating system for clusters, built on top of the Solaris operating system. It provides a single system image for a cluster that runs the Solaris™ UNIX® operating system so that it appears to the user, and to applications as a single computer. Thus, Solaris MC shows how an existing, widely used operating system can be extended to support clusters.

Most of Solaris MC consists of loadable modules extending the Solaris operating system, and minimizes modifications to the existing kernel. It provides the same message-passing interface as that in the Solaris operating system, which means that existing application and device driver binaries can run unmodified. To do this, Solaris MC has a global file system, extends process operations across all the nodes, allows transparent access to remote devices, and makes the cluster appear as a single machine on the network. It also supports remote process execution and the external network is transparently accessible from any node in the cluster.

Solaris MC is designed for high availability. If a node fails, the cluster can continue to operate. It runs a separate kernel on each node, a failed node is detected automatically and system services are reconfigured to use the remaining nodes, so only the programs that were using the resources of the failed node are affected by its failure. Solaris MC does not introduce new failure modes into the UNIX operating system.

It has a distributed caching file system with UNIX consistency semantics, based on virtual memory and file system architecture, taking advantage of the idea of using an object model as the communication mechanism, virtual memory and file system architecture, and the use of C++ as the programming language.

It has a global file system, called the Proxy File System (PXFS), which makes file accesses location-transparent. A process can open a file located anywhere on the system and each node uses the same pathname to the file. The file system also preserves the standard UNIX file access semantics, and files can be accessed simultaneously by different nodes. PXFS is built on top of the original Solaris file system by modifying the node interface, and does not require any kernel modifications.

Solaris MC has a global process management system that makes the location of a process transparent to the user. The threads of a single process must run on the same node, but each process can run on any node. The system is designed so that the semantics used in Solaris for process operations are supported, as well as providing good performance, supplying high availability, and minimizing changes to the existing kernel.

It has an I/O subsystem that makes it possible to access any device from any node on the system, no matter which device it is physically connected to. Applications can access all devices on the system as if they were local devices. Solaris MC carries over Solaris's dynamically loadable and configurable device drivers, and configurations are shared through a distributed device server.

Solaris MC has a networking subsystem that creates a single image environment for networking applications. All network connectivity is consistent for each application, no matter which node it runs on. This is achieved by using a packet filter to route packets to the proper node and performing protocol processing on that node. Network services can be replicated on multiple nodes to provide lighter throughput and lower response times. Multiple processes register themselves as servers for a particular service. The network subsystem then chooses a particular process when a service request is received. For example, rlogin, telnet, http and ftp servers are replicated on each node by default. Each new connection to these services is sent to a different node in the cluster based on a load-balancing policy, which allows the cluster to handle multiple http requests, for example, in parallel.

In conclusion, Solaris MC provides high availability in many ways, including a failure detection and membership service, an object and communication framework, a system reconfiguration service and a user-level program reconfiguration service. Some nodes can be specified to run as backup nodes, if a node attached to a disk drive fails, for example. Any failures are transparent to the user.

(2) MOSIX

MOSIX is a software package that extends the Linux kernel with cluster computing capabilities. The enhanced Linux kernel allows any sized cluster of Intel based computers to work together as a single system with SMP (symmetrical multiprocessor) architecture.

MOSIX operations are transparent to user applications. Users run applications sequentially or in parallel just as they would do on a SMP, not needing to know where their processes are running, or be concerned with what other users are doing at the time. After a new process is created, the system attempts to assign it to the best available node at that time. MOSIX continues to monitor all running processes (including the new one). In order to maximize overall cluster performance, it will automatically move processes amongst the cluster nodes when the load is unbalanced. This is all accomplished without changing the Linux interface.

As may be obvious already, existing applications do not need any modifications to run on a MOSIX cluster, nor do they need to be linked to any special libraries. In fact, it is not even necessary to specify the cluster nodes on which the application will run. MOSIX does all this automatically and transparently. In this respect, MOSIX acts like a "fork and forget" paradigm for SMP systems. Many user processes can be created at a home node, and MOSIX will assign the processes to other nodes if necessary. If the user was to run a process, it would be shown all processes owned by this user as if they were running on the node the user started them on, so providing the user with a single server image. Since MOSIX is implemented in the operating system's kernel its operations are completely transparent to user-level applications.

At its core are three algorithms: load-balancing, memory ushering, and resource management. These respond to changes in the usage of cluster resources so as to improve the overall performance of all running processes. The algorithms use preemptive (online) process migration to assign and reassign running processes amongst the nodes, which ensures that the cluster takes full advantage of the available resources. The dynamic load-balancing algorithm ensures that the load across the cluster is evenly distributed. The memory ushering algorithm prevents excessive hard disk swapping by allocating or migrating processes to nodes that have sufficient memory.

MOSIX resource management algorithms are decentralized, so every node in the cluster is both a master for locally created processes, and a server for remote (migrated) processes. The benefit of this decentralization is that processes that are running on the cluster are minimally affected when nodes are added to or removed from it. This greatly adds to the scalability and the high availability of the system.

Another advantageous feature of MOSIX is its self-tuning and monitoring algorithms. These detect the speed of the nodes, and monitor their load and available memory, as well as the interprocess communication and I/O rates of each running process. Using these data, MOSIX can make optimized decisions about where to locate processes.

In the same manner that a NFS (network file system) provides transparent access to a single consistent file system, MOSIX enhancements provide a transparent and consistent view of processes running on the cluster. To provide consistent access to the file system, it uses a shared MOSIX File System. This makes all directories and regular files throughout a cluster available from all nodes, and provides absolute consistency to files that are viewed from different nodes. The Direct File System Access (DFSA) provision extends the capability of a migrated process to perform I/O operations locally, in the current (remote) node. This provision decreases communication between I/O bound processes and their home nodes, allowing such processes to migrate with greater flexibility among the cluster's nodes, e.g., for more efficient load-balancing, or to perform parallel file and I/O operations. Currently, the MOSIX File System meets the DFSA standards.

The single system image model of MOSIX is based on the home node model. In this model, all users' processes seem to run on the users' login node. Every new process is created on the same node (or nodes) as its parent. The execution environment of a user's process is the same as the user's login node. Migrated processes interact with the user's environment through the user's home node; however, wherever possible, the migrated process uses local resources. The system is set up such that as long as the load of the user's login node remains below a threshold value, all their processes are confined to that node. When the load exceeds this value, the process migration mechanism will begin to migrate a process or processes to other nodes in the cluster. This migration is done transparently without any user intervention.

MOSIX is flexible and can be used to define different cluster types, even those on various kinds of machines or LAN speeds. Similarly to Solaris MC, MOSIX is built upon existing software so that it is easy to make the transition from a standalone to a clustered system.

It supports cluster configurations having a small or large number of computers with minimal scaling overheads. A small low-end configuration may consist of several PCs, connected by Ethernet. A larger configuration may consist of workstations connected by a higher-speed LAN such as Fast Ethernet. A high-end configuration may consist of a large number of SMP and non-SMP computers connected by a high-performance LAN, such as Gigabit Ethernet. For example, the scalable PC cluster at Hebrew University, where the MOSIX development is based, consists of Pentium servers connected by a Fast Ethernet LAN.

17.2.2 Network operating systems

Network operating systems are implementations of loosely coupled operating systems on top of loosely coupled hardware. They are software that supports the use of a network of machines and provides users, who are aware of using a set of computers, with facilities designed to ease the use of remote resources located over the network, made available as services and possibly being digital controllers, printers, microprocessors, file systems or other devices. Some resources, of which dedicated hardware devices such as printers, tape drives, etc., are classic examples, are connected to and managed by a particular machine and are made available to other machines in the network via a service or daemon. Other resources, such as disks and memory, can be organized into true distributed systems, used seamlessly by all machines. Examples of basic services available on a network operating system are starting a shell session on a remote computer (remote login), running a program on a remote machine (remote execution) and transferring files to and from remote machines (remote file transfer).

Generally speaking, network operating systems run using network routers and switches. They can be traced to three generations of development, each with distinctively different architectural and design goals.

(1) The first generation of network operating systems: monolithic architecture

Typically, first-generation network operating systems for routers and switches were proprietary images running in a flat memory space, often directly from flash memory or ROM. While supporting multiple processes for protocols, packet handling and management, they operated using a cooperative, multitasking model in which each process would run to completion or until it voluntarily relinquished the CPU. All first-generation network operating systems shared one trait: they eliminated the risks of running full-size commercial operating systems on embedded hardware. Memory management, protection and context switching were either rudimentary or non-existent, with the primary goals being a small footprint and high speed of operation. Nevertheless, these operating systems made networking commercially viable and were deployed on a wide range of products.

(2) The second-generation of network operating system: control plane modularity

The mid-1990s were marked by a significant increase in the use of data networks worldwide, which quickly challenged the capacity of existing routers and switches. By this time, it had become evident that embedded platforms could run full-size commercial operating systems, at least on high-end hardware, but with one catch: they could not sustain packet forwarding at satisfactory data rates. A breakthrough solution was needed. It came in the concept of a hard separation between the control and forwarding planes, an approach that became widely accepted after the success of the industry's first application-specific integrated circuit (ASIC)-driven routing platform, the Juniper Networks M40. These systems are free from packet switching and thus are focused on control plane functions. Unlike its first-generation counterparts, a second-generation operating system can fully use the potential of multitasking, multithreading, memory management and context manipulation, all making system-wide failures less common. Most core and edge routers or switches installed in the late-1990s are running second-generation operating systems that are currently responsible for moving the bulk of traffic on the Internet and in corporate networks.

(3) The third-generation of network operating system: flexibility, scalability and continuous operation

Although they were very successful, there are new challenges to the second generation of network operating systems. Increased competition led to the need for lower operating expenses; which made a coherent case for network software flexible enough to be redeployed in network devices across the larger part of the end-to-end packet path. From multiple-terabit routers to layer 2 (data-link) switches and security appliances, the "best-in-class" catchphrase can no longer justify a splintered operational experience. Thus, true "network" operating systems are clearly needed. Such systems must also operate continuously, so that software failures in the routing code, as well as system upgrades, do not affect the state of the network. Meeting this challenge requires availability and convergence characteristics that go far beyond the hardware redundancy that is available in second-generation routers and switches. Another key goal of third-generation operating systems is the capability to run with zero downtime (planned or unplanned).

Drawing on the lesson learned from previous designs with respect to the difficulty of moving from one operating system to another, third-generation operating systems should also make the migration path completely transparent to customers.

The following will discuss two of the most relevant topics in network operating systems.

(1) Generic kernel design

In the computer network world, both monolithic and microkernel designs can be used successfully, but the ever-growing requirements for an operating system kernel quickly turn any classic implementation into a compromise. Most notably, the capability to support a real-time forwarding plane along with state and stateless forwarding models and extensive state replication requires a mix of features not available from any existing implementation. This lack can be overcome in two ways.

Firstly, a system can be constrained to a limited class of products by design. For instance, if the operating system is not intended for mid- to low-level routing platforms, some requirements can be lifted. The same can be done for flow-based forwarding devices, such as security appliances. This artificial restriction allows the network operating system to stay closer to its general-purpose siblings at the cost of fracturing the product line-up. Different network element classes will now have to maintain their own operating systems, along with unique code bases and protocol stacks, which may negatively affect code maturity and customer experience.

Secondly, the network operating system can evolve into a specialized design that combines the architecture and advantages of multiple classic implementations.

According to the formal criteria, the JUNOS (Juniper's single network operating system) software kernel is fully customizable (Figure 17.5). At the very top is a portion of code that can be considered a microkernel, responsible for real-time packet operations and memory management, as well as interrupts and CPU resources. One level below this is a more conventional kernel that contains a scheduler, memory manager and device drivers in a package that looks more like a monolithic design. Finally, there are user-level processes in POSIX (Portable Operating System for UNIX) that actually serve the kernel and implement functions which normally reside inside the kernels of classic monolithic router operating systems. Some of these processes can be compound or run on external CPUs (or packet-forwarding engines). In JUNOS software, examples include periodic hello management, kernel state replication, and protected system domains.

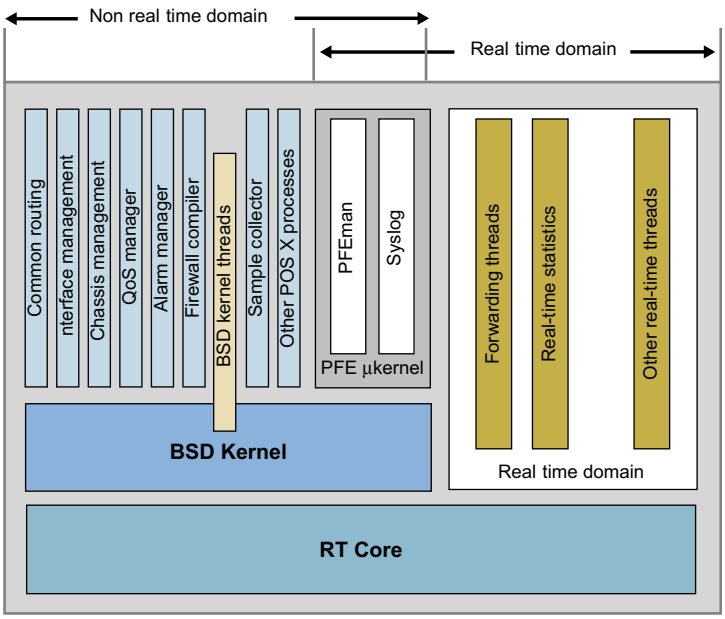


FIGURE 17.5

Generic JUNOS Software 9.0 architectural structure.

The entire structure is strictly hierarchical, with no underlying layers dependent on the operations of the top layers. This high degree of virtualization allows the JUNOS software kernel to be both fast and flexible. However, even the most advanced kernel structure is not a revenue-generating asset of the network element. Uptime is the only measurable metric of system stability and quality. This is why the fundamental difference between the JUNOS software kernel and competing designs lies in its focus on reliability.

Coupled with Juniper's industry-leading nonstop active routing and system upgrade implementation, kernel state replication acts as the cornerstone for continuous operation. In fact, the JUNOS software redundancy scheme is designed to protect data plane stability and routing protocol adjacencies at the same time. With in-service software upgrade, networks powered by JUNOS software are becoming immune to downtime related to the introduction of new features or bug fixes, enabling them to approach true continuous operation. Continuous operation demands that the integrity of the control and forwarding planes remains intact in the event of failover or system upgrades, including minor and major release changes. Devices that run JUNOS software will not miss or delay any routing updates when either a failure or a planned upgrade event occurs.

This goal of continuous operation under all circumstances and during maintenance tasks is ambitious, and it reflects Juniper's innovation and network expertise, which is unique among network vendors.

Innovation in JUNOS software does not stop at the kernel level; rather, it extends to all aspects of system operation. There are two tiers of schedulers in JUNOS software, the topmost becoming active in systems with a software data plane to ensure the real-time handling of incoming packets. It operates

in real time and ensures that QoS (quality of service) requirements are met in the forwarding path. The second-tier (non-real-time) scheduler resides in the base JUNOS software kernel and is similar to its FreeBSD counterpart. It is responsible for scheduling system and user processes in a system to enable preemptive multitasking.

In addition, a third-tier scheduler exists within some multithreaded user-level processes, where threads operate in a cooperative, multitasking model. When a compound process gets its CPU share, it may treat it like a virtual CPU, with threads taking and leaving the processor according to their execution flow and the sequence of atomic operations. This approach allows closely coupled threads to run in a cooperatively multitasking environment and avoid being entangled in extensive interprocess communication and resource-locking activities (Figure 17.6).

Another interesting aspect of multi-tiered scheduling is resource separation. Unlike first-generation designs, JUNOS software systems with a software forwarding plane cannot freeze when overloaded with data packets, as the first-level scheduler will continue granting CPU cycles to the control plane.

(2) Network routing processes

The routing protocol process daemon (RPD) is the most complex process in a JUNOS software system. It not only contains much of the actual code for routing protocols, but also has its own scheduler and

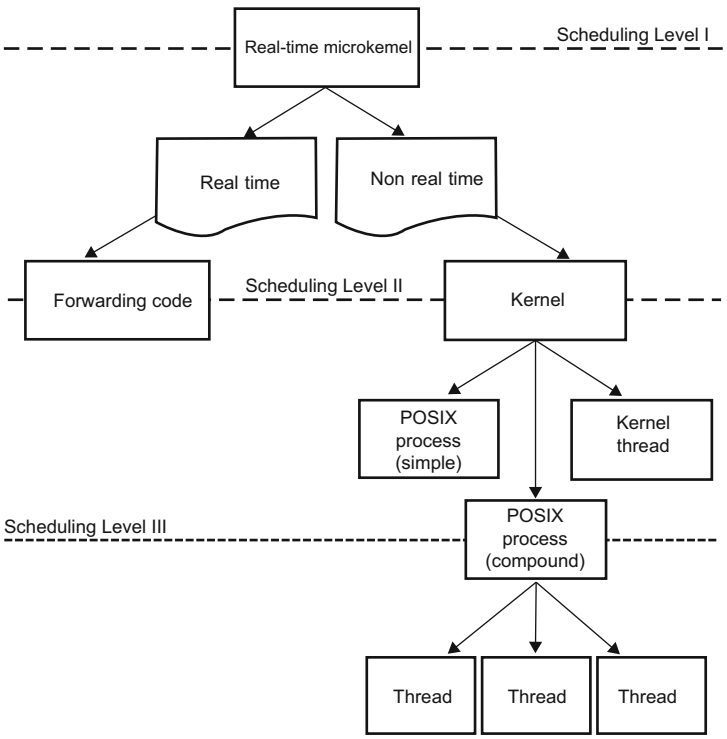


FIGURE 17.6

Multilevel CPU scheduling in JUNOS software.

memory manager. The scheduler within RPD implements a cooperative multitasking model, in which each thread is responsible for releasing the CPU after an atomic operation has been completed. This design allows several closely related threads to coexist without the overhead of interprocess communication and to scale without risk of unwanted interactions and mutual locks.

The threads within RPD are highly modular and may also run externally as standalone POSIX processes, which is how many periodic protocol operations are performed. In the early days of RPD, each protocol was responsible for its own adjacency management and control. Now, most keep-alive processing resides outside RPD, in the bidirectional forwarding detection protocol (BFD) daemon and periodic packet management process daemon (PPMD), which are, in turn, distributed between the routing engine and the line cards. The unique capability of RPD to combine preemptive and cooperative multitasking powers the most scalable routing stack in the market.

Compound processes similar to RPD are known to be very effective, but sometimes are criticized for the lack of protection between components. It has been said that a failing thread will cause the entire protocol stack to restart. Although this is a valid point, it is easy to compare the impact of this error against the performance of the alternative structure, in which every routing protocol runs in a dedicated memory space.

Assume that the router serves business VPN (virtual private network) customers, and the ultimate revenue-generating product is continuous reachability between remote sites. At the very top is a BGP (Border Gateway Protocol) process responsible for creating forwarding table entries. Those entries are ultimately programmed into a packet path ASIC for the actual header lookup and forwarding. If the BGP process hits a bug and restarts, forwarding table entries may become stale and would have to be flushed, thus disrupting customer traffic. But BGP relies on lower protocols in the stack for traffic engineering and topology information, and it will not be able to create the forwarding table without OSPF or RSVP. If any of these processes are restarted, BGP will also be affected (Figure 17.7). This case supports the benefits of running BGP, OSPF and RSVP in shared memory space, where the protocols can access common data without interprocess communication overhead.

Routing threads may operate using a cooperative, preemptive or hybrid task model, but failure recovery still calls for state restoration using external checkpoint facilities. If vital routing information were duplicated elsewhere and could be recovered promptly, the failure would be transparent to user traffic and protocol peers alike. Transparency through prompt recovery is the principal concept underlying any design for network service routines, and the main idea behind the contemporary Juniper Networks RPD implementation.

Instead of focusing on one technology or structure, Juniper Networks engineers evolve the JUNOS software protocol stack according to a "survival of the fittest" principle, toward the goal of true nonstop operation, reliability and usability. State replication, check pointing and interprocess communication are all used to reduce the impact of software and hardware failures. The JUNOS software control plane is designed to maintain speed, uptime and full state under the most unfavorable network situations.

17.2.3 Parallel operating systems

Parallelism, or concurrency, is an important requirement for large-scale numerical engineering computations, wide-area telephone networks, multi-agent database systems, and massively distributed control networks. It must be approached by both application level and kernel level in these target systems. However, application and operating system parallelism are two distinct issues. For example,

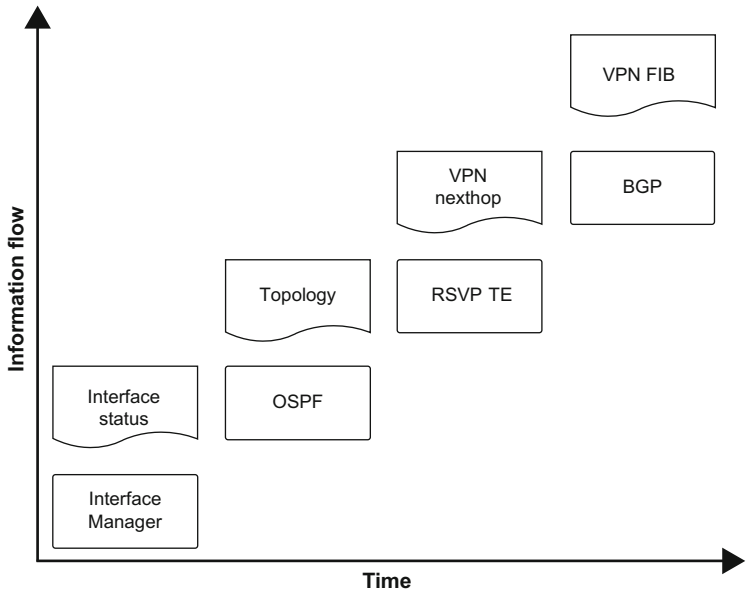


FIGURE 17.7

Hierarchical routing protocol stack operation in some network operating systems.

multiprocessor supercomputers are machines that can support parallel numerical computations, but, whether a weather forecast model can be parallel-processed on such supercomputers depends upon whether parallelism is programmed into the model. This subsection involves operating system parallelism, i.e. the parallel operating system.

Parallel operating systems are primarily used for managing the resources of parallel machines. In fact, the architecture of a parallel operating system is closely influenced by the hardware architecture of the machines it runs on. Thus, parallel operating systems can be divided into three groups: parallel operating systems for small-scale symmetric multiprocessors, parallel operating system support for large-scale distributed-memory machines, and scalable distributed shared-memory machines.

The first group includes the current versions of UNIX and Windows NT, where a single operating system manages all the resources centrally. The second group comprises both large-scale machines connected by special-purpose interconnections and networks of personal computers or workstations, where the operating system of each node locally manages its resources, and also collaborates with its peers via message passing to globally manage the machine. The third group is composed of non-uniform memory access machines where each node's operating system manages local resources and interacts at a very fine-grain level with its peers to manage and share global resources.

Parallelism required by these operating systems involves the basic mechanisms that constitute their core: process management, file system, memory management and interior communications. These mechanisms include support for multithreading, internal parallelism of the operating system components, distributed process management, parallelism systems, interprocess communication, low-level resource sharing, and fault isolation.

(1) Parallel computer architectures

Parallel operating systems in particular enable user interaction with computers with parallel architectures. The physical architecture of a computer system is therefore an important starting point for understanding the operating system that controls it. There are two famous classifications of parallel computer architectures: Flynn's and Johnson's.

(a) Flynn's classification of parallel architectures

Flynn divides computer architectures along two axes according to the number of data sources and the number of instruction sources that a computer can process simultaneously. This leads to four categories of computer architectures:

1. SISD (single instruction single data);
2. MISD (multiple instruction single data);
3. SIMD (single instruction multiple data);
4. MIMD (multiple instruction multiple data).

These four categories have been discussed in subsection 17.1.1 of this textbook.

(b) Johnson's classification of parallel architectures

Johnson's classification is oriented towards the different memory access methods. This is a much more practical approach since, as we saw above, all but the MIMD class of Flynn are either virtually extinct or never existed. We take the opportunity of presenting Johnson's categories of parallel architectures to give some examples of MIMD machines in each of those categories. Johnson divides computer architectures into:

1. UMA (Uniform Memory Access);
2. NUMA (Non-Uniform Memory Access);
3. NORMA (No Remote Memory Access).

These three categories have been also discussed in subsection 17.1.1 of this textbook.

(2) Parallel operating facilities

There are several components in an operating system that can be parallelized. Most do not approach all of them and do not support parallel applications directly. Rather, parallelism is frequently exploited by some additional software layer such as a distributed file system, distributed shared memory support or libraries and services that support particular parallel programming languages while the operating system manages concurrent task execution.

The convergence in parallel computer architectures has been accompanied by a reduction in the diversity of operating systems running on them. The current situation is that most commercially available machines run a flavor of the UNIX operating system. Others run a UNIX-based microkernel with reduced functionality to optimize the use of the CPU, such as Cray Research's UNICOS. Finally, a number of shared-memory MIMD machines run Microsoft Windows NT (soon to be superseded by the high-end variant of Windows 2000).

There are a number of core aspects to the characterization of a parallel operating system, including general features such as the degrees of coordination, coupling and transparency; and more particular aspects such as the type of process management, interprocess communication, parallelism and synchronization and the programming model.

(a) Coordination

The type of coordination among microprocessors in parallel operating systems is a distinguishing characteristic which conditions how applications can exploit the available computational nodes. Furthermore, as mentioned before, application parallelism and operating system parallelism are two distinct issues: while application concurrency can be obtained through operating system mechanisms or by a higher layer of software, concurrency in the execution of the operating system is highly dependent on the type of processor coordination imposed by the operating system and the machine's architecture. There are three basic approaches to coordinating microprocessors:

1. Separate supervisor. In a separate supervisor parallel machine, each node runs its own copy of the operating system. Parallel processing is achieved via a common process management mechanism allowing a processor to create processes and/or threads on remote machines. For example, in a multicomputer such as the Cray-T3E, each node runs its own independent copy of the operating system. Parallel processing is enabled by a concurrent programming infrastructure such as a MPI (message-passing interface) library, whereas I/O is performed by explicit requests to dedicated nodes. Having a front-end that manages I/O and that dispatches jobs to a back-end set of processors is the main motivation for a separate supervisor operating system.

2. Master-slave. Master-slave parallel operating system architecture assumes that the operating system will always be executed on the same processor, and that this processor will control all shared resources, in particular process management. A case of master-slave operating system is the CM-5. This type of coordination is particularly adapted to single-purpose machines running applications that can be broken into similar concurrent tasks. In these scenarios, central control may be maintained without any penalty to the other processors' performance since all processors tend to be beginning and ending tasks simultaneously.

3. Symmetric. Symmetric operating systems are the current most common configuration. In this type of system, any processor can execute the operating system kernel. This leads to concurrent accesses to operating system components and requires careful synchronization.

(b) Coupling and transparency

Another important characteristic of parallel operating systems is the system's degree of coupling. Just as in the case of loosely coupled hardware architectures (e.g. a network of workstation computers) and highly coupled hardware architectures (e.g. vector parallel multiprocessors), parallel operating systems can be either loosely or tightly coupled. Many current distributed operating systems have a highly modular architecture, so there is a wide spectrum of distribution and parallelism in different operating systems. To see how influential coupling is in forming the abstraction presented by a system, consider the following extreme examples: on the highly coupled end a special-purpose vector computer dedicated to one parallel application, e.g. weather forecasting, with master-slave coordination and a parallel file system; and on the loosely coupled end a network of workstations with shared resources (printer, file server) each running their own application and perhaps sharing some client-server applications. These scenarios show that truly distributed operating systems correspond to the concept of highly coupled software using loosely coupled hardware. Distributed operating systems aim at giving the user the possibility of transparently using a virtual single processor. This requires having an adequate distribution of all the layers of the operating system and providing a global unified view over process management, file system and interprocess communication, thereby allowing applications to perform transparent migration of data, computations and/or processes.

(c) Hardware vs. software

A parallel operating system provides users with an abstract computational model of the computer architecture, which can be achieved by the computer's parallel hardware architecture or by a software layer that unifies a network of processors or computers. In fact, there are implementations of every computational model in both hardware and software systems:

1. The hardware of the shared memory model is represented by symmetric multiprocessors, whereas the software is achieved by unifying the memory of a set of machines by means of a distributed shared-memory layer.
2. In the case of the distributed-memory model there are multicomputer architectures where accesses to local and remote data are explicitly different and, as we saw, have different costs. The equivalent software abstractions are explicit message-passing inter-process communication mechanisms and programming languages.
3. Finally, the SIMD computation model of massively parallel computers is mimicked by software through data parallel programming. Data parallelism is a style of programming geared towards applying parallelism to large data sets, by distributing this data over the available processors in a "divide and conquer" mode. An example of a data parallel programming language is HPF (High Performance Fortran).

(d) Protection

Parallel computers, being multiprocessing environments, require that the operating system provide protection among processes and between processes and the operating system so that erroneous or malicious programs are not able to access resources belonging to other processes. Protection is the access-control barrier, which all programs must pass before accessing operating system resources. Dual-mode operation is the most common protection mechanism in operating systems. It requires that all operations that interfere with the computer's resources and their management are performed under operating system control in what is called protected or kernel mode (as opposed to unprotected or user mode). The operations that must be performed in kernel mode are made available to applications as an operating system API (application program interface). A program wishing to enter kernel mode calls one of these system functions via an interruption mechanism, whose hardware implementation varies among different processors. This allows the operating system to verify the validity and authorization of the request and to execute it safely and correctly using kernel functions, which are trusted and well-behaved.

17.3 DISTRIBUTED AND PARALLEL FACILITIES

Distributed operating systems should provide some facilities for implementing distributed and parallel operations. This section introduces five of these facilities: process migration, parallel virtual machine (PVM), remote procedure call (RPC), distributed file systems (DFS), and message-passing interface (MPI). In this section, UNIX is the example used for the explanation and description of these facilities.

17.3.1 Process migration

As we have learnt, a "process" is one of the basic elements in computer operating systems. A process consists of data, a stack, register contents, and the state specific to the underlying operating system,

such as parameters related to process, memory, and file management. A process can have one or more threads of control. Threads, also called lightweight processes, consist of their own stack and register contents, but share a process's address space and some of the operating-system-specific state, such as signals. The task concept was introduced in the last chapter as a generalization of the process concept, whereby a process can be defined as being decoupled into a task and a number of threads. A traditional process is represented by a task with one thread of control.

Process migration is the act of transferring a process between two computers or machines (the source and the destination nodes) during its execution. Some of architectures also define a host or home node, which is the node where the process logically runs. Figure 17.8 shows a high-level view of process migration. The transferred state includes the process's address space, execution point (register contents), communication state (e.g., open files and message channels) and other operating-system-dependent states. Task migration represents transferring a task between two machines during execution of its threads.

During migration, two instances of the migrating process exist: the source instance is the original process, and the destination instance is the new process created on the destination node. After migration, the destination instance becomes a migrated process. In systems with a home node, a process that is running on other machines may be called a remote process (from the perspective of the home node) or a foreign process (from the perspective of the hosting node).

(1) Implementation

There are many issues to consider when developing a system that will migrate an active process from one node (the source) to another (the destination). Most of these issues must consider the interactions between two factors, the first being the level of transparency between the source and destination. For example, do the two nodes share the same view of the file system? Or, do they have the same hardware peripherals? The second factor is the management of the process state.

For each element of state that defines a process, the implementation of a migration scheme has three options when migrating the process to a destination node:

(a) move the state from source to destination node along with the process, and let the destination node service requests that use the state;

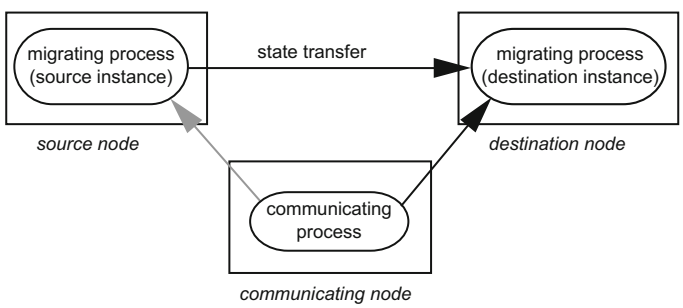


FIGURE 17.8

Process migration consists of extracting the state of the process on the source node, transferring it to the destination node where a new instance of the process is created, and updating the connections with other processes on communicating nodes.

(b) keep the state on the source node, and have the process send requests to use that state from the destination node back to the source node;
(c) ignore the state, and accept the resulting change in the semantics of the program on the destination node with respect to the source node.

Let us now illustrate the difference between these decisions with a design example. Each process has a set of file descriptors that describe aspects of files that the process is currently accessing; for example, the current access position within the file, and (implicitly) the name of the file being accessed. When migrating a process to a destination node, we might choose to forward this file state information to the destination node along with the process. Later, when the process running on the destination node attempts to access a file, the destination node can locally fulfill that request by using the state information imported from the source node.

Alternatively, the state of the file can be kept on the source node. When the process running on the destination node attempts to access a file, it forwards the request back to the source node. The source node fulfills the request and returns the results (for example, data read from the file) to the destination node.

This example also serves to illustrate the relationship between the process state and transparency between the nodes. Forwarding the state to the destination node is only useful if transparency between the nodes gives us a guarantee that the state will have same meaning on the destination node as it did on the source. For example, forwarding the name of an open file and current access position to a destination node is of no use if this node does not have the same view of the file system as the source does.

Our open-file example illustrates the interplay between the decision of where to store a process's state and the level of transparency available on the system. There are a number of other decisions similar to the open-file decision that must be made as well. These are explored in many related sources, along with the decisions made by the designers of several working migration systems.

(2) Algorithms

Although there are many different migration implementations and designs, most of them can be summarized in the following steps (see also Figure 17.9):

1. A migration request is issued to a remote node. After negotiation, migration is accepted.
2. A process is detached from its source node by suspending its execution, declaring it to be in a migrating state, and temporarily redirecting communication as described in the following step.
3. Communication is temporarily redirected by queuing up arriving messages directed to the migrated process, and by delivering them to the process after migration. This step continues in parallel with steps 4, 5, and 6, as long as there are additional incoming messages. Once the communication channels are enabled after migration (as a result of step 7), the migrated process is known to the external world.
4. The process state is extracted, including memory contents, processor state (register contents), communication state (e.g., opened files and message channels), and relevant kernel context. The communication state and kernel context depend upon the operating system. Some internal states in the local operating system are not transferable. The process state is typically retained on the source node until the end of migration, and in some systems it remains there even after migration completes. Processor dependencies, such as register and stack contents, have to be eliminated in the case of heterogeneous migration.

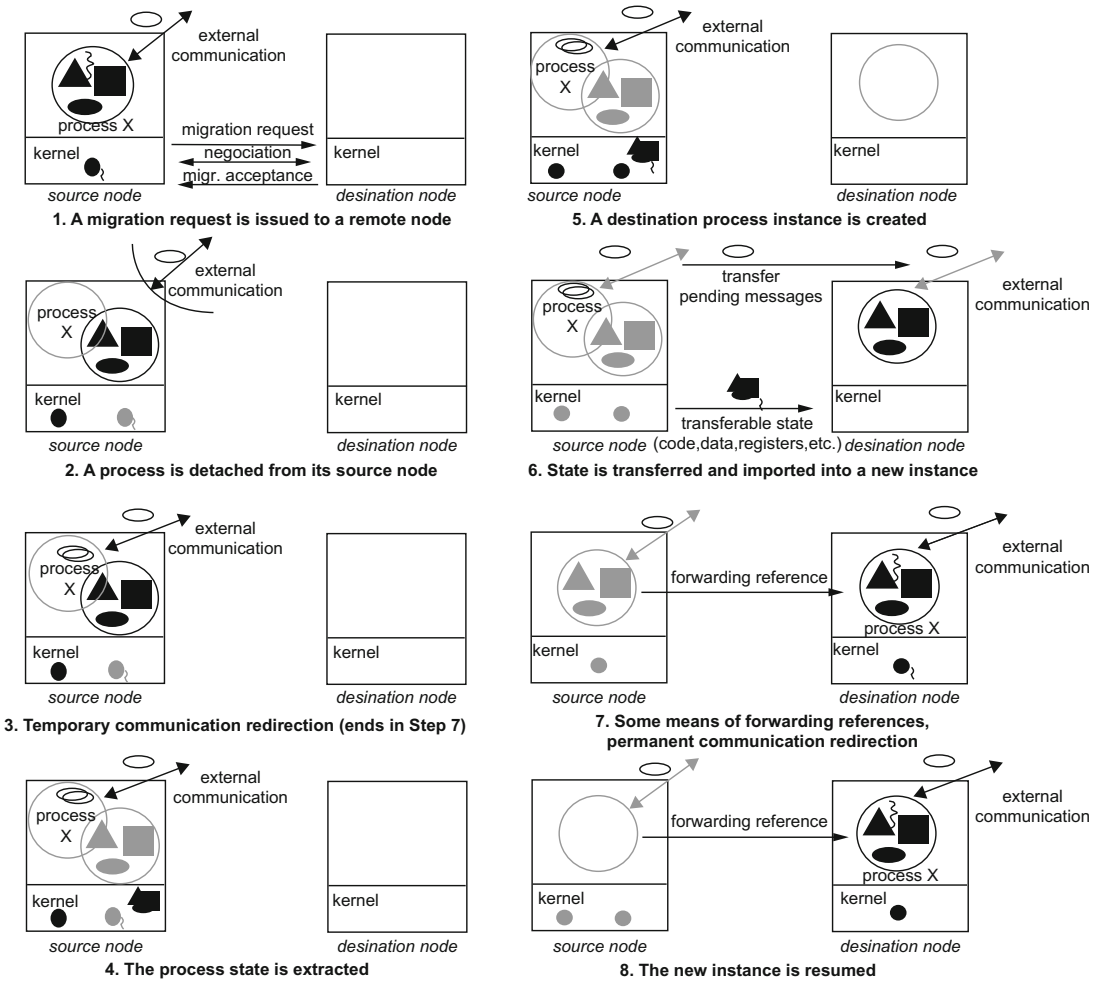


FIGURE 17.9

Migration algorithm. Many details have been simplified, such as kernel migration, when the process is actually suspended, when the state is transferred, and how messages are transferred etc. These details vary subject to the particular implementation scheme and design.

5. A destination process instance is created into which the transferred state will be imported. It is not activated until a sufficient amount of state has been transferred from the source process instance. After that, the destination instance will be promoted to being a regular process.
6. State is transferred and imported into a new instance on the remote node. Not all of the state needs to be transferred; some parts could be lazily brought over after migration is completed.
7. Some means of forwarding references to the migrated process must be maintained. This is required in order to communicate with the process or to control it. It can be achieved by registering the

current location at the home node, by searching for the migrated process, or by forwarding messages across all visited nodes. This step also enables migrated communication channels at the destination and it ends step 3 as communication is permanently redirected.

8. The new instance is resumed when sufficient state has been transferred and imported. With this step, process migration completes. Once all of the state has been transferred from the original instance, it may be deleted on the source node.

17.3.2 Parallel virtual machine (PVM)

Parallel virtual machine (PVM) is an integrated set of software tools and libraries that emulates a general-purpose, flexible, heterogeneous concurrent computing framework on interconnected computers of varied architecture. The overall objective of the PVM system is to enable such a collection of computers to be used cooperatively for concurrent or parallel computation. Under PVM, a user-defined collection of serial, parallel, and vector computers appears as one large distributed-memory computer. Thus, large computational problems can be solved by using the aggregate power of many smaller machines.

Owing to its ubiquitous nature (specifically, the virtual machine concept) and also because of its simple but complete programming interface, the PVM system has gained widespread acceptance in the high-performance, scientific computing, community.

(1) PVM system

PVM is composed of two parts. The first is a daemon, called pvmd3 and sometimes abbreviated pvmd, and the second is a library of PVM interface routines.

Pvmd3 is designed so any user with a valid login can install it on on a machine. When a user wishes to run a PVM application, he or she first creates a virtual machine by starting up PVM. The application can then be started from, for example, a UNIX prompt on any of the hosts. Multiple users can configure overlapping virtual machines, and each can execute several PVM applications simultaneously.

The library of PVM interface routines contains a functionally complete repertoire of primitives that are needed for cooperation between tasks of an application. This library contains user-callable routines for message passing, spawning processes, coordinating tasks, and modifying the virtual machine.

The PVM computing model is based on the notion that an application consists of several tasks, each being responsible for a part of the application's computational workload. Sometimes an application is parallelized along its functions; that is, each task performs a different function, for example, input, problem setup, solution, output, and display. This process is often called functional parallelism. A more common method of parallelizing an application is called data parallelism. In this method all the tasks are the same, but each one only knows and works with small part of the data. This is also referred to as the SIMD (single instruction and multiple data) model of computing. PVM supports either, or a mixture of these methods. Depending on their functions, tasks may execute in parallel and may need to synchronize or exchange data, although this is not always the case. An example diagram of the PVM computing model is shown in Figure 17.10 and an architectural view of the PVM system, highlighting the heterogeneity of the computing platforms supported, is shown in Figure 17.11.

Tasks access PVM resources through a library of standard interface routines. These routines allow the initiation and termination of tasks across the network as well as communication and synchronization between tasks. PVM tasks may possess arbitrary control and dependency structures. In other

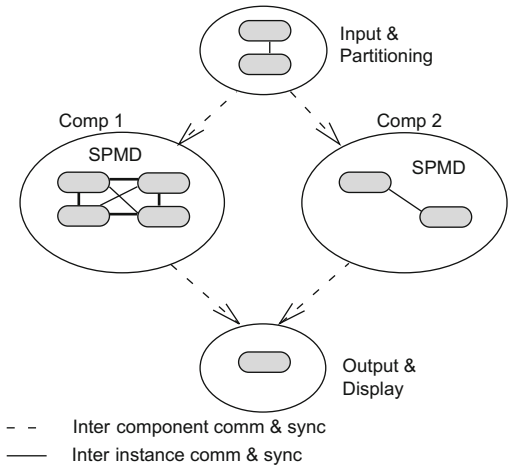


FIGURE 17.10

PVM (parallel virtual machine) computation model.

words, at any point in the execution of a concurrent application, any task in existence may start or stop other tasks or add or delete computers from the virtual machine. The PVM message-passing primitives are oriented towards heterogeneous operation, involving strongly typed constructs for buffering and transmission. Communication constructs include those for sending and receiving data structures as well as high-level primitives such as broadcast, barrier synchronization, and global summation.

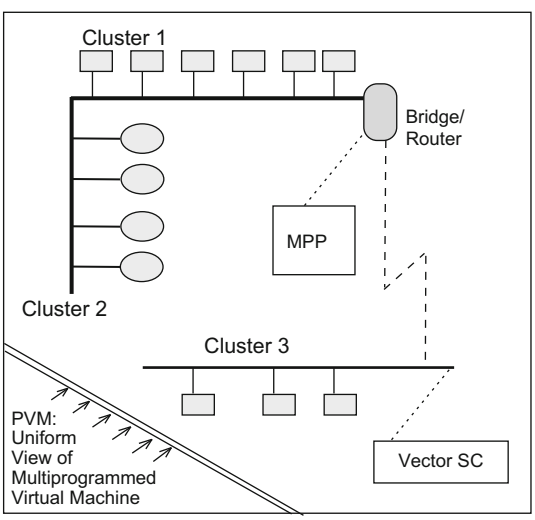


FIGURE 17.11

PVM (parallel virtual machine) architectural model.

(2) Using PVM

Once PVM is running, an application using PVM routines can be started from a UNIX command prompt on any of the hosts in the virtual machine. Therefore, an application program need not be started on the same machine the user happens to start PVM. Below are two important functions, message passing and dynamic process group, provided in PVM with UNIX, and an example program written in C that illustrates how PVM with UNIX can perform a master-slave model with communication between slaves.

(a) Message passing in PVM with UNIX

Sending a message is composed of three steps. Firstly, a send buffer must be initialized by a call to pvm initsend or pvm mkbuf. Secondly, the message must be "packed" into this buffer using any number and combination of pvm pk routines. Finally, the completed message is sent to another process by calling the pvm send routine or multicast with the pvm mcast routine. In addition there are collective communication functions that operate over an entire group of tasks, for example, broadcast and scatter and gather.

A message is received by calling either a blocking or non-blocking receive routine and then "unpacking" each of the packed items from the receive buffer. The receive routines can be set to accept any message, or any message from a specified source, or any message with a specified message tag, or only messages with a given message tag from a given source. There is also a probe function that indicates whether a message has arrived, but does not actually receive it.

Message passing in PVM is operated by means of the message-passing interface (MPI) which is explained in subsection 17.5.

(b) Dynamic process group in PVM with UNIX

The dynamic process group functions are built on top of the core PVM routines. There is a separate library, libgpvm3.a, that must be linked with user programs that make use of any of the group functions. The pvmd does not perform the group functions, this is handled by a group server that is automatically started when the first group function is invoked. There is some debate about how groups should be handled in a message-passing interface, as there are efficiency and reliability issues. There are trade-offs between static versus dynamic groups, and it could be argued that only tasks in a group should call group functions.

In PVM, the group functions are designed to be very general and transparent to the user at some cost in efficiency. Any PVM task can join or leave any group at any time without having to inform any other task in the affected groups. Tasks can broadcast messages to groups of which they are not a member, and any PVM task may call any of the group functions at any time. The exceptions are pvm lvgroup, pvm barrier, and pvm reduce, which by their nature require the calling task to be a member of the specified group.

These routines allow a task to join or leave a user-named group. The first call to pvm joingroup creates a group with name group and puts the calling task in this group. pvm joingroup returns the instance number of the process in this group. Instance numbers run from 0 to the number of group members minus 1.

To assist the user in maintaining a contiguous set of instance numbers despite joining and leaving, the pvm lvgroup function does not return until the task is confirmed to have left. A pvm joingroup

called after this return will assign the vacant instance number to the new task. If several tasks leave a group and no tasks join, then there will be gaps in the instance numbers.

A program written in C with UNIX is given in Figure17.12 and is an example of a master-slave model with communication between slaves. PVM is not restricted to this model for example, any PVM task can initiate processes on other machines. A master-slave model is, however, a useful programming paradigm and is simple to illustrate. The master calls pvm mytid, which, as the first PVM call, enrols this task in the PVM system. It then calls pvm spawn to execute a given number of slave programs on other machines in PVM. The master program contains an example of broadcasting messages in PVM. It broadcasts to the slaves the number of slaves started and a list of all the slave tides. Each slave program calls pvm mytid to determine their task ID in the virtual machine, and then uses the data broadcast from the master to create a unique ordering from 0 to n proc minus 1. Subsequently, pvm send and pvm recv are used to pass messages between processes. When finished, all PVM programs call pvm exit to allow PVM to disconnect any sockets to the processes, flush I/O buffers, and to allow PVM to keep track of which processes are running.

17.3.3 Remote procedure call (RPC)

Remote procedure call (RPC) is a facility in distributed operating systems that allows programs to make procedure calls on other machines across a network. The idea of the RPC facility is based on a local procedure call. The local procedure is executed by the same process as the program containing the procedure call. The call is performed in the following manner: (1) The caller places arguments passed to the procedure in some well-specified place, usually on a stack. (2) Control is transferred to the sequence of instructions which constitute the body of the procedure. (3) The procedure is executed. (4) After completion, return values are placed in a well-specified place and control returns to the calling point.

A remote procedure is executed by a different process, which usually works on a different machine. This corresponds to the client-server model, in which the caller is the client and the callee is the server. First, the client sends a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply. The arguments and results are transmitted through the network along with the request and reply, respectively. The call of a remote procedure should be transparent so that the programmer cannot see much difference between a local and a remote procedure. This is ensured by code responsible for the communication between the server and the client called a stub (a client stub and a server stub; see Figure 17.13), which hides network communication mechanisms from the programmer.

The client stub is a local procedure, which is called by the client when requesting remote procedure execution. Its task is to send to the server a request and arguments passed by the client, then receive results, and pass them to the client. The task of the server stub is to receive the request and the arguments sent by the client stub, call (locally) the proper procedure with the arguments, and send the results of execution back to the client stub. Before the connection between server and client is established, the client must locate the server on a remote host. To this end, when the server starts, it registers itself with a binding agent, i.e. registers its own name or identifier and a port number at which it is waiting for requests. When the client calls a remote procedure provided by the server, it first asks the binding agent on the remote host for the port number at which the server is waiting (listening), and then sends the request for a remote procedure execution to that port (see Figure 17.14).

```
#include "pvm3.h"
#define SLAVENAME "slavel"

main( ) {
     int mytid;  / * my task id  */
     int tids[32];  /* slave task ids  */
     int n, nproc, i, who, msgtype;
     float data[100], result[32];

     /* enroll in pvm  */
     mytid = pvm_mytid( );
     /* start up slave tasks  */
     puts ("How many slave programs  (1-32) ?  ");
     scanf ("%d", &nproc);
     pvm_spawn (SLAVENAME,  (char**) 0,  0,  '' '' , nproc, tids);
     /* begin user program */
     n = 100;
     initialize_data ( data, n );

     /* broadcast initial data to slave tasks */
     pvm_initsend (PvmDataRaw);
     pvm_pkint (&nproc, 1, 1);
     pvm_pkint (tids,  nproc, 1);
     pvm_pkint (&n, 1, 1);
     pvm_pkfloat (data, n, 1);
     pvm_mcast(tids, nproc, 0);

     /* wait for results from slaves  */
     msgtype = 5;
     for ( i=0 ; i<nproc ; i++)  {
          pvm_recv( -1, msgtype );
          pvm_upkint( &who, 1, 1  );
          pvm_upkfloat ( &result[who], 1, 1 );
          printf ("I got %f from %d\n", result[who],who);
     }

     /* program finished exit PVM before stopping */
     pvm_exit ();
}
```

FIGURE 17.12

A program (written in C) as an example of a master-slave model with communication between slaves in PVM (parallel virtual machine).

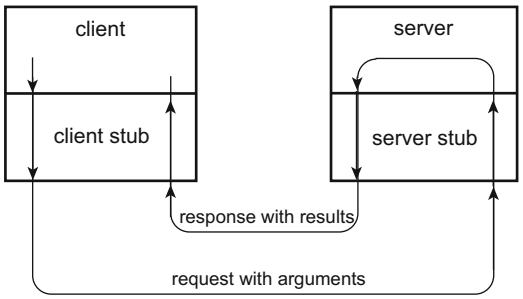


FIGURE 17.13

The communication model of RPC (remote procedure call).

The following gives a brief explanation of the RPC developed by Sun Microsystems, which is specific for UNIX and written in C/C++.

(1) External data representation (XDR)

As mentioned above, the server and the client of the RPC may work on different machines with differing architectures, which raises the problem of data conversion. Therefore, along with RPC, Sun Microsystems has proposed a solution to the problem called external data representation (XDR), which consists of streams (XDR streams) and filters (XDR filters). Filters are procedures used to convert data between the native format and the standard XDR format (Table 17.1). This conversion is called serialization, and the reverse conversion from XDR format to the native one is called deserialization. Data converted to XDR format are stored in XDR streams.

It is possible to construct a filter for any abstract data-type-like structure, linked list and so on. To simplify construction of such filters an XDR language has been proposed by Sun Microsystems. This allows us to describe complex data types, so that a special tool, RPC generator (rpcgen), can generate XDR filters for them. Thus, the RPC language is an extension of the XDR language. It is worth noting that rpcgen supports type declarations, but does not support variable declarations. Thus, the types can be used as formal parameter types and return value types of remote procedures, or as components of more complex types.

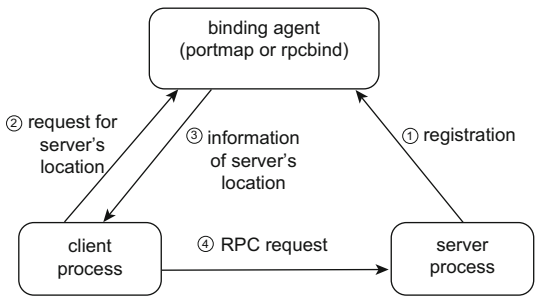


FIGURE 17.14

Locating a RPC (remote procedure call) server.

Table 17.1 XDR (External Data Representation) Filters

	Filters	Description
Primitive filters	xdr char, xdr u char, xdr int, xdr u int, xdr long, xdr u long, xdr short, xdr u short, xdr float, xdr double	Conversion of simple types
	xdr enum, xdr bool	Conversion of enumeration types
	xdr void	No data conversion (empty filter)
Compound filters	xdr string, xdr byte, xdr array, xdr vector	Conversion of arrays
	xdr opaque	Serialization or deserialization of data without format change
	xdr union	Conversion of discriminated union
	xdr reference	Conversion of pointers

(2) Building RPC

Let us start the exploration of the RPC technique from a centralized application consisting of a main program and a number of subprograms (functions or procedures). Building a distributed application consists in converting some of the subprograms into remote procedures. RPC stubs mean that modification of the body of the main program and the body of the subprograms is avoided. For this purpose, a special tool, rpcgen, is used to create the stubs.

Depending on compile-time flags, rpcgen generates a code for the client and server stubs, a header file for basic constants, code for data conversion (XDR), the client and the server-side templates, and the makefile template. To generate this, rpcgen requires a protocol specification written in the RPC language. First of all, this specification contains remote procedure declarations (i.e. specification of arguments and return values). One server can provide several remote procedures, which are grouped in versions. Each procedure is assigned a number, which is unique within the version. The version is also assigned a number, which is unique within the server program. The server program has also a number, which should be distinct from all server numbers on the machine. Thus, a remote procedure is identified by the program number, the version number and the procedure number. All the numbers are long integers. The program numbers (in hexadecimal) are grouped in Table 17.2.

(3) Using RPC

RPC can be used for building remote services. To request a remote service it is necessary to write a client program, while the server program has been written by someone else. Even if the server has

Table 17.2 Program Numbers of RPC (Remote Procedure Call)

Decimal	Hexadecimal	Description
0 —	1FFFFFFF	Defined by Sun
20000000 —	3FFFFFFF	Defined by user
40000000 —	5FFFFFFF	User-defined for programs that dynamically allocate numbers
60000000 —	FFFFFFFF	Reserved for future use

been created by means of rpcgen, the protocol specification in the RPC language may not be available for the programmer writing the client program. Therefore, use of rpcgen to construct the client program is complex. Moreover, rpcgen can generate only a sample client program (a skeleton), which in practise requires great modification.

Hence, to request a remote service, the rpc call routine from the RPC library is used. This allows a procedure to be called, identified by program number, version number, and process number on the machine identified by host. The argument inproc is the XDR filter used to encode the procedure parameters, and similarly outproc is used to decode the results. The argument in is the address of the procedure parameter(s), and out is the address of where to place the result(s). If a procedure takes several parameters, it is necessary to define a structure containing the parameters as its members. It is also necessary to define or generate the XDR filter for the structure. The argument, network type, defines a class of transports which is to be used.

17.3.4 Distributed file systems (DFS)

In computers, data are traditionally stored in units known as files in a hierarchical tree where the nodes are known as directories (note: here the node is not a computer or a machine). File systems are an abstraction that enables users to read, manipulate and organize data in a uniform view, independent of the underlying storage devices, which can range between anything from floppy drives to hard drives and flash memory cards.

As computer networks started to evolve in the 1980s it became evident that the old file systems had many limitations that made them unsuitable for multiuser environments. At first, many users started to use FTP (File Transfer Protocol) to share files. Although this method avoided the physical movement of removable media, files still needed to be copied twice: once from the source computer onto a server, and a second time from the server onto the destination computer. Additionally users had to know the physical addresses of every computer involved in the file-sharing process. Computer companies, for example Sun Microsystems, tried to solve these problems. Entirely new systems, such as Network File System and Parallel File System, were developed to add new features, such as file locking, to existing file systems. These new systems were defined as distributed file systems (DFS) and provide the same interface as the old file systems to processes.

Transparency is one of the basic standards of DFS, which includes access transparency, location transparency, scaling transparency, and replication transparency. Access transparency allows current stand-alone systems to be migrated to networks and old programs do not need to be modified. Location transparency makes it possible for the clients to access the file system without knowledge of where the physical files are located. For example, this makes it possible to retain paths even when the servers are physically moved or replaced. Scaling transparency is the ability to incrementally grow the system without having to make changes to its structure or its applications. The larger the system, the more important is scaling transparency.

One possibility in distributed systems is to spread the physical storage of data to many locations, called data or file replication. Data replication, which makes multiple copies, increases the availability of the data by making it possible to share the load. In addition, it enhances reliability by enabling clients to access other locations if one has failed. One important requirement for replication is to provide replication transparency; users should not need to be aware that their data are stored in multiple locations. Another use for distributed storage is the ability to disconnect a computer from the

network but continue working on files. When reconnected, the system would automatically propagate the necessary changes into the other locations.

In a multiple-user environment it is desirable to only allow access to data to authorized users, because most DFS require data protection or access security. Users must be authenticated and only requests to files where users have access right should be granted. Most DFS implement some kind of access control lists. Additionally protection mechanisms may include protection of protocol messages by signing them with digital signatures and encrypting the data.

(1) Schemes and semantics

As one of the components of distributed operating systems, DFS is based on fundamental schemes and semantics which are independent of the concrete operating systems and distributed system types.

(a) Naming schemes

There are three main approaches to naming schemes in a DFS. In the simplest approach, files are named by some combination of their host name and a local name, which guarantees a unique name over the system. This scheme is neither location-transparent nor location-independent, but the same file operations can be used for both local and remote files. In this first approach, component units remain isolated, although means are provided to refer to a remote file. The second approach, popularized by Sun Microsystems, provides means for individual machines to attach (or mount, in UNIX jargon) remote directories to their local name spaces. Once a remote directory is attached locally, its files can be named in a location-transparent manner. The resulting name structure is versatile; usually it is a forest of trees in UNIX, one for each machine, with some overlapping (i.e., shared) sub-trees. A prominent property of this scheme is the fact that the shared name space may not be identical on all the machines. However, the scheme has the potential for creating customized name spaces for individual machines.

Total integration between the component file systems is achieved using the third approach: a single global name structure that spans all the files in the system. Consequently, the same name space is visible to all clients. Ideally, the file system structure should be isomorphic to the structure of a conventional file system. In practice, all important criteria for evaluating the above naming structures produce administrative complexity. The most complex and difficult to maintain is the NFS structure. The effects of a failed machine, or taking a machine offline, are to make some arbitrary set of directories become unavailable. Likewise, migrating files from one machine to another requires changes in the name spaces of all the affected machines. In addition, a separate accreditation mechanism had to be devised for controlling which machine is allowed to attach which directory to its name space.

(b) Sharing semantics

The semantics for file sharing are important criteria for evaluating any file system that allows multiple clients to share files. In particular, these semantics should specify when modifications of data by a client are observable, if at all, by remote clients. For the following discussion we need to assume that a series of file accesses (i.e., reads and writes) attempted by a client to the same file are always enclosed between the open and close operations. We denote such a series of accesses as a file session.

1. UNIX semantics. These semantics include two aspects. In the first, every read of a file sees the effects of all previous writes performed on that file in the DFS. In particular, writes to an open file by

a client are visible immediately by other (possibly remote) clients who have this file open at the same time. In the second, it is possible for clients to share the pointer of the current location in the file. Thus, the advancing of the pointer by one client affects all sharing clients.

2. Session semantics. Session semantics also comprises two rules. Firstly, writes to an open file are visible immediately to local clients, but invisible to remote clients who have the same file open simultaneously. Secondly, once a file is closed, the changes made to it are visible only in later-starting sessions. Instances of the file that are already open do not reflect these changes.

3. Immutable shared files semantics. A different, quite unique approach is that of immutable shared files. Once a file is declared as shared by its creator, it cannot be modified any more. An immutable file has two important properties: its name may not be reused, and its contents may not be altered. Thus, the name of an immutable file signifies the fixed contents of the file, not the file as a container for variable information. The implementation of these semantics in a distributed system is simple since the sharing is in read-only mode.

4. Transaction-like semantics. Identifying a file session with a transaction yields the following semantics: The effects of file sessions on a file and their output are equivalent to the effect and output of executing the same sessions in some serial order. It implements these semantics by locking a file for the duration of a session. Please refer to the rich literature on database management systems to understand the concepts of file transactions and file locking. For example, in the Cambridge File Server, the beginning and end of a transaction are implicit in the open file, close file operations, and transactions can involve only one file. Thus, a file session in that system is actually a transaction.

(c) File replication

As mentioned earlier, replication of files is a useful redundancy for improving availability. The basic requirement for a replication scheme is that different replicas of the same file reside on failure-independent machines, so the availability of one replica is not affected by the availability of the others. This requirement obviously implies that replication management is inherently a location-dependent activity. Provisions for placing a replica on a particular machine must be available. Replication control includes determining the degree of replication and placement of replicas. Under certain circumstances, it is desirable to expose these details to users. The existence of replicas should be invisible to higher levels. At some level, however, the replicas must be distinguished from one another by having different lower-level names. This can be accomplished by first mapping a file name to an entity that is able to differentiate the replicas.

(d) Remote-access methods

Consider a client process that requests to access (i.e., read or write) a remote file. Assuming the server storing the file was located by the naming scheme, the actual data transfer should take place. There are two complementary methods for handling this type of data transfer.

1. Remote service. Requests for accesses are delivered to the server, which performs the accesses, and the results are forwarded back to the client. There is a direct correspondence between accesses and traffic to and from the server. Access requests are translated to messages for the servers, and server replies are packed as messages sent back to the clients. Every access is handled by the server and results in network traffic. For example, a read corresponds to a request message sent to the server and a reply to the client with the requested data. A similar notion called remote open is defined in some literatures.

2. Caching. If the data needed to satisfy the access request are not present locally, a copy of those data is brought from the server to the client. Usually the amount of data brought over is much larger than the data actually requested (e.g., whole files or pages versus a few blocks). Accesses are performed on the cached copy in the client side. The idea is to retain recently accessed disk blocks in cache so repeated accesses to the same information can be handled locally, without additional network traffic. Caching performs best when the stream of file accesses exhibits locality of reference. A replacement policy (e.g. least recently used) is used to keep the cache size bounded. There is no direct correspondence between accesses and traffic to the server. Files are still identified, with one master copy residing at the server machine, but copies of (parts of) the file are scattered in different caches. When a cached copy is modified, the changes need to be reflected in the master copy and, depending on the relevant sharing semantics, in any other cached copies. A write access may incur substantial overhead. The problem of keeping the cached copies consistent with the master file is referred to as the cache consistency problem; please refer to related papers and books for further details.

(2) Application examples

There are two types of distributed file systems, network file systems and parallel file systems, which have widespread application.

(a) Network file systems

NFS, the network file system, is probably the most prominent network service using RPC. It allows files on remote hosts to be accessed in exactly the same way as a user would access any local files. This is made possible by a mixture of kernel functionality on the client side (which uses the remote file system) and a NFS server on the server side that provides the file data. This file access is completely transparent to the client, and works across a variety of server and host architectures. When someone accesses a file over NFS, the kernel places a RPC call to the NFS daemon on the server machine. This call takes the file handle, the name of the file to be accessed, and both the user's and group's identifications as parameters. These are used in determining access rights to the specified file. In order to prevent unauthorized users from reading or modifying files, user and group identifications must be the same on both hosts. On most implementations, the NFS functionality of both client and server are implemented as kernel-level daemons that are started from user space at system boot. These are the NFS daemon on the server host, and the block I/O daemon running on the client host. To improve throughput, the block I/O daemon performs asynchronous I/O using read-ahead and write-behind; also, several NFS daemons are usually run concurrently.

(b) Parallel file system

Parallel file systems address the performance problem faced by current network client-server file systems: the read and write bandwidth for a single file is limited by the performance of a single server (memory bandwidth, processor speed, etc.). Parallel applications suffer from this performance problem because they present I/O loads equivalent to traditional supercomputers, which are not handled by current file servers. If users resort to parallel computers in search of fast execution, the file systems of parallel computers must also aim at serving I/O request rapidly. Dividing a file among several disks enables an application to access that file quicker by issuing simultaneous requests to several disks. This technique is called declustering.

If the file blocks are divided in a round-robin way among the available disks it is called striping. In a striped file system, either individual files are striped across the servers (per-file striping), or all the

data are striped, independently of the files to which they belong, across the servers (per-client striping). In the first case, only large files benefit, but in the second, given that each client forms its new data for all files into a sequential log, even small files benefit. With per-client striping, servers are used efficiently, regardless of file size. Striping a large amount of writes allows them to be done in parallel. On the other hand, small writes will be batched.

For the parallel file systems applied in clusters, a few computers are connected to storage, known as I/O computers, which serve data to the rest of the nodes of the cluster. In a cluster environment, large files are shared across multiple computers, making a parallel file system well suited for I/O subsystems. Generally, a parallel file system includes a metadata server, which contains information about the data on the I/O computers. Metadata are the information about a file, for example, its name, location, and owner. Some parallel file systems use a dedicated machine as the metadata server, while other parallel file systems distribute this functionality across the I/O computers.

The parallel virtual file system has been developed as an open-source system for Linux-based clusters. It is very easy to install and compatible with existing binaries. This system creates files over the underlying file system on I/O computers using traditional read, write, and map commands. The metadata server in a parallel virtual file system can be a dedicated computer or one of the I/O computers or clients. The node serving as the metadata server runs a daemon which manages the metadata for the files in the file system. In a parallel virtual file system, data are distributed across multiple I/O computers. The metadata server provides information about how the data are distributed and maintains the locks on the distributed files for shared access. I/O computers run a daemon that stores and retrieves files on local disks of the I/O computers.

17.3.5 Message-passing interface (MPI)

The message passing interface (MPI) is intended to be a standard interface for message passing for applications running on distributed-memory concurrent computers and computer networks. MPI is also useful in building libraries of mathematical software for such machines. It was designed to easily allow heterogeneous implementations and virtual communication channels.

MPI is not specifically designed for use by parallelizing compilers, and does not provide support for active messages, not does it contain any support for fault tolerance, and provides reliable communications (or fails the program). It is a message-passing interface, not a complete parallel computing programming environment, thus, issues such as parallel I/O, parallel program composition, and debugging are not addressed. Finally, MPI provides no explicit support for multitasking (and multithreading), although one of its design goals was to ensure that it can be implemented efficiently for any multitask (and multithread) environment.

(1) MPI concepts

MPI includes concepts and routines. Both the point-to-point and the collective communication routines depend heavily on the approaches taken to groups and contexts, and to a lesser extent on process topologies. It is necessary to explain the concepts of group and context, which are, in turn, bound together into abstract communicator objects.

(a) Process group

A process group is an ordered collection of processes, within which each process is uniquely identified by its rank within the ordering. For a group of n processes the ranks run from 0 to n−1. This definition

of groups closely conforms to current practice. Process groups can be used in two important ways; they can be used to specify which processes are involved in a collective communication operation, such as a broadcast, and they can also be used to introduce task parallelism into an application, so that different groups perform different tasks. If this is done by loading different executable codes into each group, then we refer to this as MIMD task parallelism. Alternatively, if each group executes a different conditional branch within the same executable code, then we refer to this as SIMD task parallelism (also known as control parallelism).

(b) Communication contexts

Communication contexts were initially proposed to allow the creation of distinct, separable message streams between processes, with each stream having a unique context. A common use of contexts is to ensure that messages sent in one phase of an application are not incorrectly intercepted by another phase. The point here is that the two phases may actually be calls to two different third-party library routines, and the application developer has no way of knowing whether the message tag, group, and rank completely disambiguate the message traffic of the different libraries from one another and from the rest of the application. Context provides an additional criterion for message selection, and hence permits the construction of independent message tag spaces.

(c) Communicator objects

A communication operation is specified by the communication context used, and the group, or groups, involved. In a collective communication, or in a point-to-point communication between members of the same group, only one group needs to be specified, and the source and destination processes are given by their rank within this group. In a point-to-point communication between processes in different groups, two groups must be specified. In this case the source and destination processes are given by their ranks within their respective groups. In MPI, abstract objects called "communicators" are used to define the scope of a communication operation. Communicators used in intra-group and inter-group communication are referred to as intra- and inter-communicators, respectively. An intra-communicator can be regarded as binding together a context and a group, while an inter-communicator binds together a context and two groups, one of which contains the source and the other the destination. Communicator objects are passed to all point-to-point and collective communication routines to specify the context and the group, or groups, involved in the communication operation.

(d) Application topologies

In many applications the processes are arranged with a particular topology, such as a two- or three-dimensional grid. MPI provides support for general application topologies that are specified by a graph in which processes that communicate are connected by an arc. As a convenience, MPI provides explicit support for four-dimensional Cartesian grids. For a Cartesian grid, either periodic or aperiodic boundary conditions may apply in any specified grid dimension. In MPI, a group either has a Cartesian or graph topology, or no topology.

(e) User-defined datatypes

MPI provides mechanisms to specify general, mixed (of different types), non-contiguous message buffers. This is done by allowing the user to define the datatype (which consists of a set of types and memory offsets) by using MPI datatype-definition routines. Once the datatype is defined, it can be

passed into any of the point-to-point or collective communication routines. The effect of this will be for data to be collected out of possibly non-contiguous memory locations, transmitted, and then placed into possibly non-contiguous memory locations at the receiving end. It is up to the implementation to decide whether the data of a general datatype should be first packed in a contiguous buffer before being transmitted, or whether they can be collected directly from where they reside. User-defined datatypes as supported by MPI allow the convenient and (potentially) efficient transmittal of general array sections (in Fortran 90 terminology), and arrays of (sub-portions of) records or structures. Since all send and receive routines specify numbers of data items of a particular type, whether built-in or user-defined, implementations have enough information to provide translations that allow an MPI program to run on heterogeneous networks.

(2) Point-to-point communication

MPI provides for point-to-point communication, with message selectivity explicitly based on source process, message tag, and communication context. The source and tag may be wild-carded, so that in effect they are ignored in message selection. The context may not be wild-carded.

In the point-to-point communication model, the source and destination processes are specified by means of a group and a rank. For intra-group communication the group and context are bound together in an intra-communicator. For inter-group communication the groups containing the source and destination processes are bound together with the context in an inter-communicator. Thus, a send routine is passed a handle to a communicator object, the rank of the destination process, and the message type to fully specify the context and destination of a message. A receive routine uses the same three things to determine message selectivity.

A send operation can take place in one of three communication modes:

1. Standard mode: a message sent in standard mode does not require a corresponding receive to have been previously posted on the destination process. Such a message will still be delivered when this receive is posted some time later.
2. Ready mode: a message sent in ready mode requires that receive has been previously posted on the destination process. If this is not the case the outcome is indeterminate. In standard mode, the send can return before the matching receive has been posted.
3. Synchronous mode: for a message sent in synchronous mode the send operation does not return until a matching receive has been posted on the destination process.

For each of these three communication modes, a send operation can either be locally blocking or non-blocking, so there are a total of six different types of send routine. A blocking send routine will not return until the data locations specified in the send can be safely reused without corrupting the message. A non-blocking send routine does not wait for any particular event to occur before returning. Instead it returns a handle to a communication object that can subsequently be used when calling routines that check for completion of the send operation.

A receive operation may also be locally blocking or non-blocking and either of these two types of receive may be used to match any of the six types of send. A blocking receive will not return until the message has been stored at the locations indicated by this receive. A non-blocking receive returns a handle to a communication object, and does not wait for any particular event to occur. The handle can be used subsequently to check the status of the receive operation, or to block until it completes. A non-blocking receive also returns a handle to a "return status object" which is used to store the length,

source, and tag of the message. When this receive has completed this information can then be queried by calling an appropriate routine.

(3) Collective communication

Collective communication routines provide for coordinated communication among a group of processes. The process group and context are given by the intra-communicator object that is input to the routine. The MPI collective communication routines have been designed so that their syntax and semantics are consistent with those of the point-to-point routines. In addition, the collective communication routines may be, but do not have to be, implemented using the MPI point-to-point routines.

Collective communication routines do not have a tag argument, but must be called by all members of the group with consistent arguments. As soon as a process has completed its role in the collective communication it may continue with other tasks. Thus, a collective communication is not necessarily barrier synchronization for the group. On the other hand, an MPI implementation is free to have barriers inside collective communication functions. In short, the user must program as if the collective communication routines do have barriers, but cannot depend on any synchronization from them. MPI does not include non-blocking forms of the collective communication routines.

In MPI collective communication routines are divided into two broad classes: data movement routines, and global computation routines.

(a) Collective data movement routines

There are three basic types of collective data movement routine: broadcast, scatter, and gather. There are two versions of each of these. In the one-all case, data are communicated between one process and all others; in the all-all case, data are communicated between each process and all others. The all-all broadcast, and both varieties of the scatter and gather routines, involve each process sending distinct data to each process, and/or receiving distinct data from each process. All processes must send and/or receive buffers of the same type and length. The one-all broadcast routine broadcasts data from one process to all other processes in the group. The all-all broadcast broadcasts data from each process to all others, and on completion each has received the same data. Thus, each process ends up with the same output buffer, which is the concatenation of the input buffers of all processes, in rank order. The one-all scatter routine sends distinct data from one process to all processes in the group. This is also known as "one-to-all personalized communication". In the all-all scatter routine, each process scatters distinct data to all processes in the group, so the processes receive different data from each process. This is also known as "all-to-all personalized communication". The communication patterns in the gather routines are the same as in the scatter routines, except that the direction of flow of data is reversed. In the one-all gather routine one process (the root) receives data from every process in the group. The root process receives the concatenation of the input buffers of all processes, in rank order. The all-all gather routine is identical to the all-all scatter routine.

(b) Global computation routines

There are two basic global computation routines in MPI: reduce and scan. Both reduce and scan routines require the specification of an input function. One version is provided in which the user selects the function from a predefined list, and in the second version the user supplies (a pointer to) a function. Thus, MPI contains four reduce and four scan routines.

Problems

1. Please explain what heavyweight processes, middleweight processes, and lightweight processes are in multiprocessor scheduling; analyze their advantages and problems; and give the solutions of these problems.
2. Why is "dynamic scheduling" better than "static scheduling" in multiprocessor scheduling policies?
3. Please analyze these multiprocessor scheduling policies: single shared ready queue, co scheduling, round robin scheduling, dynamic partitioning, and hand off scheduling. Then check whether these scheduling policies satisfy real time standards.
4. Can you plot several diagrams to give the steps of the "continuous algorithm" in multiprocessor operating systems?
5. Explain the differences of the mutual exclusion mechanism for multiprocessor synchronization from that for single processor synchronization.
6. Based on Figure 17.9, please write every step of process migration.
7. Based on the program in Figure 17.12, explain why, under PVM, a user defined collection of serial, parallel and vector computers appears as if it is one large distributed memory computer.
8. Based on Table 17.2, analyze how the RPC uses "program number" to identify a remote procedure.
9. Research the Sun Microsystems Solaris operating system, and find which of the naming schemes given in this textbook is used in this Solaris operating system.
10. Please work out all the differences between the message passing interface in Chapter 17 and the message queue in Chapter 16 of this textbook.

Further Reading

Randy Chow, Theodore Johnson. Distributed Operating Systems and Algorithm Analysis. Addison Wesley / Prentice Hall. 1997.

Andrew S. Tanenbaum. Modern Operating Systems (3rd Edition). Prentice Hall. 2007.

Felician Aleu. Operating Systems for Parallel Processing. Studies in Information and Control, Volume 13, No. 2, PP 115 118. 2004.

Sundararajan Sriram, Shuvra S. Bhattacharyya. Embedded Multiprocessors Scheduling and Synchronization. CRC Press. 2000.

Bodhisattwa Mukherjee et al. A Survey of Multiprocessor Operating System Kernels. Georgia Institute of Technology: GIT CC 92/05. 1993.

Russell Clarke, Benjamin Lee. Cluster Operating Systems. Monash University, Australia. 2000.

Sun Microsystems (http://www.sun.com). Solaris operating systems. http://www.sun.com/software/solaris. Accessed: November 2008.

Juniper Networks (http://www.juniper.net). White paper: Network Operating System Evolution. 2006.

Poznan University of Technology. Distributed Computing Systems Laboratory Exercises. December 2008.

Joao Garcia et al. IV. Parallel Operating Systems. IST/INESC, Portugal. 2008.

Eitan Frachtenberg et al. Design Parallel Operating Systems via Parallel Programming. Los Alamos National Laboratory. 2007.

Al Geist et al. PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge. 1994.

Eliezer Levy, Abraham Silberschatz. Distributed File Systems: Concepts and Examples. ACM Computing Survey, 4(22), 321 374. 1990.

AFS Frequently Asked Questions. http://www.angelfire.com/hi/plutonic/afs faq.html. Accessed: December 2008.

Steven Zenith. MPI: A Message Passing Interface. The MPI Forum. http://www.mpi forum.org/. Accessed: December 2008.

The Amoeba Distributed Operating System. http://www.cs.vu.nl/pub/amoeba/amoeba.html. Accessed: December 2008.

CHAPTER

Industrial control system operation routines

18

The operability, correctness, and reliability of technical processes are essential qualities in industrial control systems. They are important to both process and systems during design, start-up, operation, maintenance and repair. To ensure a process is working properly, it is necessary to perform state monitoring, device testing, component installation, system configuration, instrument calibration, and fault diagnosis.

To do these functions, an industrial control system requires software and hardware known as system routines, which are the subject of this chapter.

The routines are implemented in system firmware, which is a combination of software and hardware. The concept of "firmware" has evolved to mean almost any programmable content of a hardware device, not just machine code, but also configurations and data for application-specific integrated circuits (ASICs), programmable logic devices, etc. Examples include programmable read-only memory (PROM), a microprocessor chipset's PCI bus, all I/O interfaces and buses, and most peripherals.

Typically, firmware involves very basic, low-level operations, without which an electronic device would not work. It is the low-level nature of firmware that makes it able to support system routines. Therefore, it is strongly suggested that readers should study it, since a good understanding of firmware will help in understanding the system routines in this chapter.

18.1 SELF-TEST ROUTINES

After it is switched on a control system requires time to establish itself because: (1) the system needs to recognize all the components and devices and their respective attributes and parameters; (2) the different parts of the system need to understand each other with regard to functions and statuses; (3) the system needs to ensure that it has no mechanical or electronic fault, and no software error.

When shut down, the control system changes from its working state to an idle state before the power is cut off, so as to avoid potential damage to mechanical parts, electronic hardware, and software. The shutdown procedure needs some time because: (1) before the power is cut off, all the microprocessor-unit boards or chipsets in the system need to withdraw from their current process by saving the program data and changing the program mode; (2) all the hardware devices in the system must terminate their movements and activities; (3) the system needs to ensure that the shutdown process does not produce any mechanical or electronic fault, and no software error.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10018-X

To manage both booting and shutdown procedures, an embedded system therefore needs to have a self-test functionality. Most use a package of special firmware, known as self-test routines, for this purpose. This section will first discuss power-on (booting) and then power-down (shutdown) self-test routines.

Since the 1990s, an innovative electronic and semiconductor technique, known as system-on-chip (SoC), has been developed. This means integrating all components of a computer onto a single integrated circuit (chip), including all digital, analog, mixed-signal, and often radio frequency functions. Its typical application is in the area of embedded systems. Testing these embedded processors in the SoC architecture has always been a challenge because most traditional testing techniques do not work. Although still under development, the self-test routines for SoC architected devices will be introduced in subsection 4, because they will definitely replace existing technologies to lead a production revolution in electronic and semiconductor manufacturing.

18.1.1 Basic input/output system (BIOS)

In general, the basic input/output system (BIOS) is a concept comprising both hardware and software. Its hardware aspect is the firmware, which means all programmable content of the hardware components and devices. Its software aspect is the set of machine instructions used to boot the system chipset, termed boot program or boot code. The boot programs of the BIOS are typically stored in the EEPROM, sometimes called Flash. In a PC it is stored in CMOS; nonvolatile memory on the motherboard. It was traditionally called CMOS RAM because it used a low-power CMOS SRAM powered by a small battery when system power was off.

When a control system is switched on, the BIOS is the first thing executed by the system. The motherboard begins the power-on self-test (POST) process. The typical steps in booting a PC are as follows:

1. When the computer's power is first turned on, the CPU initializes itself, as triggered by a series of clock ticks (pulses) generated by the system clock. Part of this initialization is to look to the BIOS in the ROM for its first instruction in the start-up program. This will be the instruction to run the POST, in a predetermined memory address. In its turn, the POST begins by checking the chip's PCI bus array and then tests the CMOS RAM. If no battery failure is detected, it then continues to initialize the CPU, and checks the inventoried hardware devices (such as video card, AGP, etc.), secondary storage devices, such as a hard drives and floppy drives, ports and other hardware devices, such as keyboard and mouse, to ensure they are functioning properly.
2. Once the POST has determined that all components are functioning properly and the CPU has successfully initialized, the BIOS looks for an operating system to load, typically looking to the CMOS for its location. In most PCs, the operating system loads from the hard drive. The order of drives that the CMOS looks to in order to locate the operating system is called the boot sequence, which can be changed by altering its setup. Looking to the appropriate boot drive, the BIOS will first encounter the boot record, which tells it where to find the beginning of the program files of operating system and the subsequent program file that will initialize it.
3. Once the operating system initializes, the BIOS copies its files into memory and the operating system basically takes control of the boot process. Now in control, the operating system

performs another inventory of the system's memory and memory availability (which the BIOS has already checked) and loads the device drivers needed to control the peripherals. This is the final stage in the boot process, after which the user can access the system's applications.

On most current computer motherboards, the peripheral connecting interface (PCI) bus arrays are used as the main trunks linking their CPU's internal buses with other types of bus such as ISA, SCSI, RS-series, AGP, and so on. Bus systems on motherboards have several bridges, that are ASIC device, to undertake these transactions. It is these complex bus arrays that the motherboards and other microprocessor chipsets rely on to perform the POST, which has required the BIOS code to become more complex in its turn.

18.1.2 System booting self-test routines

The self-test of system hardware and software in the boot stage once power is turned on is a necessary operation for an industrial control system. Without it, the system cannot be properly built up to carry on the control activities loaded by users. POST, meaning power-on self-test, is the conventional term to describe this operation, but it can be used for the hardware and software tests applied after completing the boot process. For example, some industrial control systems issue self-test in the diagnostic routines (see the next section for the diagnostic routines), or in the synchronization stage after the boot process completes. Although these tests can perform the same operations as the POST does in the boot process, they are not the same as the POST in a strict technical sense.

The boot programs, or the BIOS of a microprocessor unit carry on this POST. In most industrial control systems, they are stored in an EEPROM (also called Flash) so that they are editable and adjustable. When a microprocessor unit is first powered up, its program counter is pointed at the beginning of the boot program code in the EEPROM, hence the boot code first runs and the POST first implements. As we know, when the booting process completes, the boot program is immediately moved to the operating system code of the application programs.

POST is the main task during the boot process, performing two kinds of task: hardware initialization and self-diagnostics. The following lists the work performed by POST routines in systems with only one microprocessor:

1. Initializing and testing the internal buses of a microprocessor-unit. As described in section 5.1.4, the internal bus system of a microprocessor includes three buses: the address bus, the data bus, and the control bus. Some registers resident in the chipset control these buses. Their initialization involves writing to these registers to set up their states and to configure their functionality and physical parameters. Self-test can be performed during the execution of these writes, and will stop if the buses have hardware faults.
2. Initializing and testing the internal I/O ports of a microprocessor-unit. As described in section 5.1.3, there are input ports and output ports in a microprocessor unit, which are ASICs containing some registers. Initialization requires two steps: (1) read from these ports to detect whether or not they exist; and (2) write to these registers to initialize the ASICs. If there are hardware faults, these writes are stopped after the fault is recorded in the microprocessor-unit registers.

3. Initializing and testing the buses of a microprocessor-unit board. As described in section 5.1 and section 6.3, a microprocessor-unit board should have other bus systems to connect with peripheral devices. The bus system of a microprocessor-unit board is complicated in topography, consisting of bridges and arbiters that actually are ASICs consisting of some registers. The initialization of this bus system needs to write to these registers to set up the ASICs. Self-test operations are performed during the execution of these writes and will be stopped if they have hardware faults.
4. Initializing and testing the peripheral devices. A microprocessor-unit board may connect with some of the peripheral devices listed in section 6.3. They communicate with the microprocessor by means of the bus system. Their initialization takes three steps: (1) assign the memory-mapped I/O registers to each device; (2) read their controllers to verify their existence; and (3) write the work and configuration parameters to the registers. During these steps, initialization stops if these devices do not exist or have hardware faults.
5. Initializing and testing the memory of the microprocessor unit. The memory inside a microprocessor unit is SDRAM. Firstly, geometry of the SDRAM bank is determined, and then its controller is tested. Its geometry is probed by writing some values into various locations in SDRAM, then reading back these values and working out where the holes in the memory map are. The microprocessor is set in protected mode and some loops are executed to ensure that every place in the SDRAM has been probed. The SDRAM controller is initialized with the relevant registers inside the microprocessor unit. The memory bank map includes the number of banks, the starting and ending addresses of each bank, and so on.
6. Initializing the interrupt controller of a microprocessor unit. This generates the interrupt vector table which contains all kinds of possible interrupts and their handlers. The following are necessary to build up the interrupt vector table for a microprocessor: (1) for each possible interrupt and its handler, setting up the stack structure of its descriptor; (2) for each possible interrupt and its handler, setting up the stack structure of its opcode; and (3) for each possible interrupt, setting up the stack structure that can be used to store the seven last stack frames.

For a control system or control device with more than one microprocessor unit, after its power switch is turned on, the following procedure starts immediately: (1) each microprocessor-unit board or chipset boots itself independently; (2) after being successfully booted, they communicate with each other to synchronize; (3) the main, or mother, microprocessor-unit board processes the installation and configuration for the system; (4) when this is complete, the system is ready to undertake control functionality.

18.1.3 System shutdown self-test routines

There are two ways to implement the shutdown operation of an industrial control system:

1. Hard shutdown: It is called “hard power off” if the power is immediately eliminated from everywhere in the system without allowing any devices to make preparation once a power-down request is made. This could damage the system; possibly both the hardware and the software in the control system could be broken.

2. Soft shutdown; Also called "soft power down" if every device in the system is prepared before the power supply is cut off to the system. This could protect an industrial control system from damage to both its hardware and its software.

Soft shutdown is recommended owing to its obvious safety benefits. It asks for the system to run a preparation process between the power-down request and when the power supply is cut off. A typical shutdown process follows the procedure listed below:

1. The power-down request is first signaled to the system either by pressing its system on/off button or by clicking such a shutdown button in its human machine interface. This request, as a signal or as an interrupt, first arrives at the microprocessor unit located on the board or chipset that connects with this system on/off button. This microprocessor-unit board can be called the power control MPU board.
2. After the power control MPU board receives this signal, it normally sends a "power down request" message to the motherboard of the system to start the power-down process. The motherboard will coordinate the power-down process thereafter. For some control systems, when receiving this power-down request message, the motherboard issues a "power is shutting down" display as a pop-up window in the human machine interfaces, to ask users for confirmation. This confirmation needs to be sent to the motherboard microprocessor.
3. After the shutdown request is confirmed, all the microprocessor-unit boards and devices of the control system start the preparation for powering down.

(a) Motherboard:

(i) its operating system initializes the system power-down manager to coordinate the power-down process;
(ii) the system power-down manager, after it is initialized, sends power-down message to all the connected boards and devices;
(iii) the operating system of the motherboard then changes its own mode from normal to idle; it then reports to the system power-down manager;
(iv) after the power-down manager has received reports from all devices, including itself, that their system modes have changed to idle, the system power-down manager sends a "turn off power" message to the microprocessor of the power control MPU board.

(b) Power control MPU board:

(i) its operating system sends a power-down request message to all connected devices;
(ii) it then changes its own system modes from normal to idle; once it and all devices have made the mode change, it reports completion of the system mode change to the power-down manager;
(iii) on receiving the "turn off power" message from the motherboard, it sends a "power supply off" command to the power supply control device.

(c) Microprocessor-unit boards other than the motherboard or the power control MPU board:

(i) it sends a power-down request message to all connected devices;
(ii) it then changes its own system mode from normal to idle; once this is done, it reports completion of the mode change to its own motherboard (which may not be the motherboard of the control system).

(d) Power supply control device: this device implements only one operation: on receiving the power supply off command from the power control MPU board, it cuts the power resource off removing the power completely.

In such a power-down process, the change of system mode is the key operation required for each microprocessor-unit board or chipset. The sequence of steps is: (1) stop the currently running application processes; (2) save all the necessary working data; (3) save all the necessary data of the processes waiting in the queue; (4) save all the memory (RAM, DRAM, etc.) contents; (5) save the reason for the power-down if required; and (6) enter idle mode and run the idle task in the operating system programs.

18.1.4 System-on-chip device self-test routines

System-on-chip, or SoC, is where several subsystems are placed on a single semiconductor chip, and may be used to describe many of today's complex application-specific integrated circuits (ASICs), where many functions previously spread across multiple chips are now provided by just one. Microprocessor and microcontroller cores, digital signal processors (or DSPs; a DSP is a specialized microprocessor with optimized architecture for fast operations needed by digital signal processing such as audio and speech signal processing, sensor array processing, and digital image processing, etc.), memory blocks, communications cores, sound and video cores, radio frequency (RF) cells, power management, and high-speed interfaces may all be part of one SoC.

In recent years, there have been great advances in the speed, capability, and complexity of integrated circuits, such as ASIC chips, random-access-memory (RAM) chips, and microprocessor chips, making possible the development of SoC devices. Figure 18.1 gives an example of a SoC design for embedded real-time control applications. In this figure, the proposed architecture aims at a single chip solution for most common high-performing embedded real-time control systems. It integrates all the digital electronics into a single FPGA (field-programmable gate array), reducing the external logic to a minimum.

The SoC technology was roughly generated in the later years of the twentieth century, and is still developing. Test designs of SoC devices have involved sophisticated theoretical issues, and complicated electronic and semiconductor technologies. It actually has become impossible to provide a comprehensive and detailed discussion of SoC test methods in this subsection. We will briefly discuss the key aspect of SoC test methods: the self-test for embedded processors and programmable cores in SoC. Other aspects including the SoC test and self-test on intellectual-property cores and global interconnect components can be obtained from the existing and forthcoming literature.

(1) Embedded processors in SoC architecture

Within a SoC architecture, each of the embedded processors should accomplish at least the following two tasks: (1) realize a large portion of the system's functionality in the form of embedded code routines to be executed by the processor(s); (2) control and synchronize the exchange of data among the different intellectual-property cores of the SoC.

The first of these offers high flexibility to the SoC designers because they can use the processor's inherent programmability to efficiently update, improve and revise the system's functionality just by adding or modifying existing embedded software (code and data) stored in embedded memory cores.

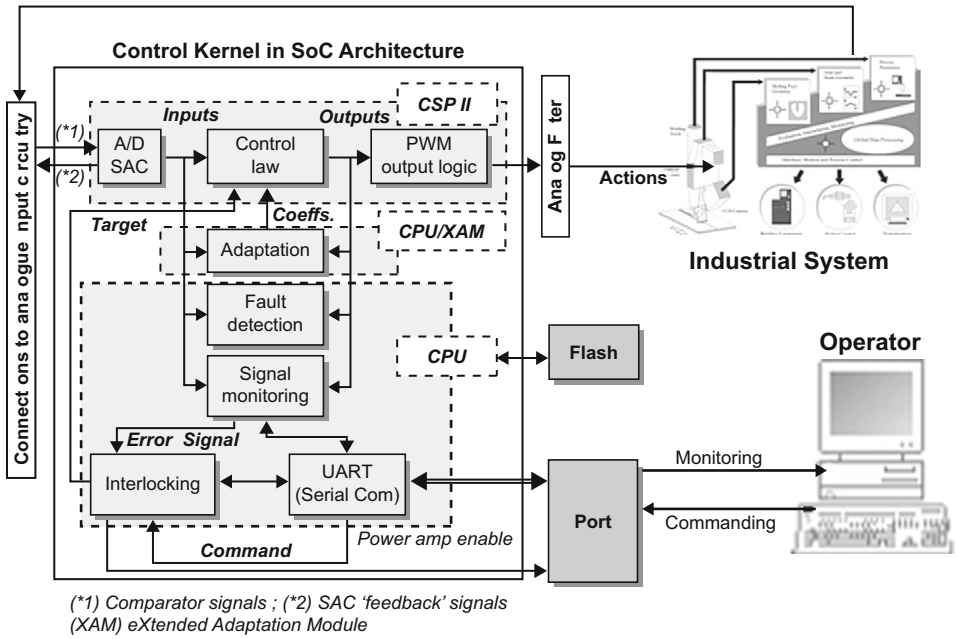


FIGURE 18.1

The system-on-chip (SoC) architecture of a real-time industrial control system.

The second task offers excellent accessibility and communication from the processor to all internal cores of the SoC; the embedded processor can therefore be used for several reliability-related functions.

At present, the most common arrangement is to have two embedded processors in a SoC. For example, an embedded microprocessor unit or a digital signal processor (DSP) can be used for the main processing functionality, while a DSP can take over the heavier data processing for specialized signal-processing algorithms. In architectures where the SoC communicates with several external data channels, a separate embedded processor associated with its dedicated memory subsystem may deal with each of the communication channels, while another can be used to coordinate the flow of data.

A typical SoC architecture containing representative types of core is shown in Figure 18.2. Here, the SoC architecture consists of a number of embedded processor cores, with one playing a central role and others having specialized tasks. Several memory cores (dynamical or static RAM, ROM, etc) can also be used, as shown in Figure 18.2, each dedicated to a different task. Other cores are used to interface between the SoC and its output and yet other are used to interface with the analog world, converting analog to digital and vice versa.

The intellectual-property cores are divided into three categories, depending on the level of optimization, and flexibility of reuse: soft cores, firm cores, and hard cores. A soft core is usually a synthesizable hardware-description-language specification that can be retargeted to various semiconductor processes. A firm core is generally specified as a gate-level netlist, suitable for placement and routing, while a hard core also includes technology-dependent information such as

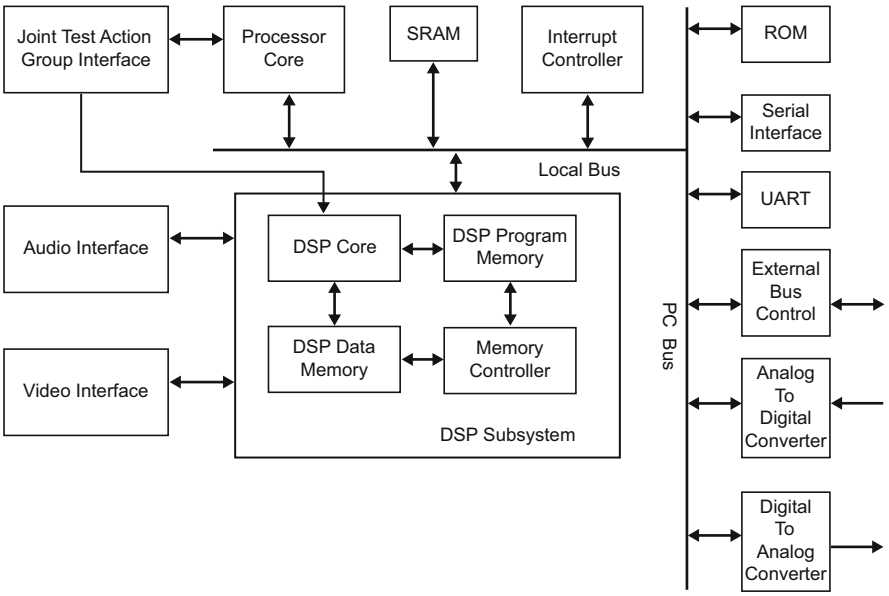


FIGURE 18.2

Typical embedded processor in system-on-chip architecture.

layout and timing. Generally, soft cores provide the highest flexibility, allowing many core parameters to be changed prior to synthesis, while hard cores provide little or none. Some cores are patented and copyrighted, while others are freely available.

A tremendous variety of intellectual-property cores are available, so designers can select from a rich pool of well-designed and carefully verified cores and can integrate them, in a plug-and-play fashion, in the system they are designing. An idea of the variety of types of intellectual-property cores that a SoC may contain is also given in Figure 18.3.

Testing of both microprocessors and embedded processors in a SoC has always been a challenge because most traditional testing techniques fail when applied to them. This is due to the complex sequential structure of processor architectures, which consists of high-performance data-path units and sophisticated control logic for performance optimization. Structured design-for-testability and hardware-based self-testing techniques, which usually have a non-trivial impact on a circuit's performance, size and power, cannot be applied without serious consideration and careful incorporation into the processor design.

(2) Hardware-based embedded processor self-test

Hardware-based self-testing or built-in self-testing (BIST) techniques were proposed several decades ago. This uses the circuit itself generate the test patterns instead of requiring the application of external patterns. Even more appealing is a technique where the circuit itself decides whether the obtained results are correct, since it is usually necessary to insert extra circuitry for the generation and analysis of patterns. The general format of a built-in self-test design is illustrated in Figure 18.4. It contains

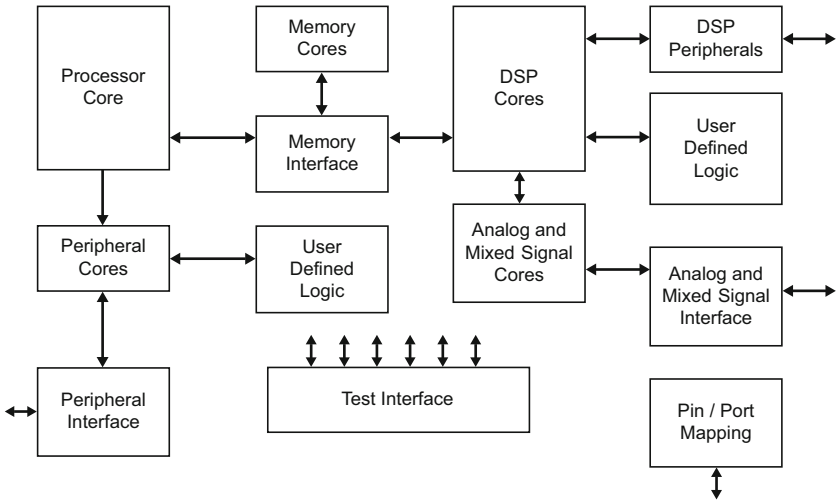


FIGURE 18.3

A core types of system-on-chip architecture.

a means for supplying test patterns to the device-under-test and also a means of comparing the device's response to a known correct sequence.

There are many ways to generate the stimuli in Figure 18.4. Most widely used are the exhaustive and the random approaches. In the exhaustive approach, the test length is 2^N where N is the number of inputs to the circuit. The exhaustive nature of the test means that all detectable faults will indeed be detected, given the space of the available input signals. An N-bit counter is a good example of an exhaustive pattern generator, but for circuits with large values of N, the time needed to go through the complete input space might be prohibitive. An alternative approach is to use random testing, which involves the application of a randomly chosen subset of the 2^N possible input patterns selected such that a reasonable fault coverage is obtained. An example of a pseudorandom pattern generator is the linear-feedback-shift-register (or LFSR), which is shown in Figure 18.5(a). A LFSR consists of

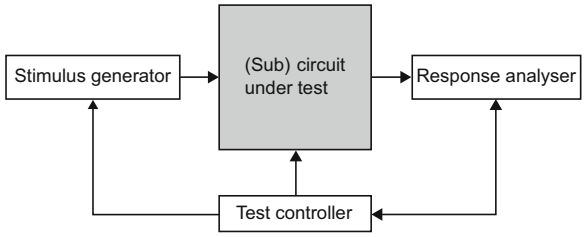


FIGURE 18.4

General format of built-in self-test (BIST) structure for self testing the embedded processors in system-on-chip architecture.

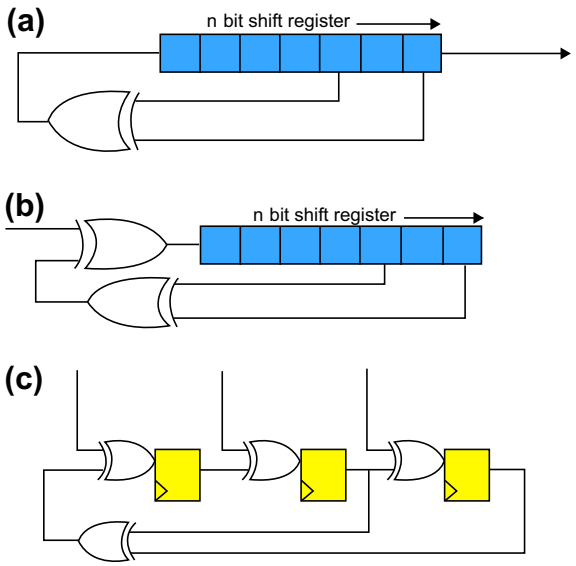


FIGURE 18.5

Three pseudorandom patterns for generating stimuli: (a) an N-bit linear-feedback-shift-register (LFSR); (b) an N-bit single-input-signature-register (SISR); (c) a 3-bit multiple-input-signature-register (MISR).

a serial connection of 1-bit registers. An N-bit LFSR cycles through $2^N - 1$ states before repeating the sequence, which produces a seemingly random pattern. Initialization of the registers to give a specified seed value determines what will be generated, sequentially.

The response analyzer could be implemented as a comparison between the generated and the expected response stored in an on-chip memory, but this approach has too high an area overhead, and so it is impractical. A cheaper technique is to compress the responses before comparing them, to minimize the amount of on-chip memory needed. The response analyzer then consists of circuitry that dynamically compresses the output of the circuit-under-test and a comparator. The compressed output is often called the signature of the circuit, and the overall approach is called signature analysis.

A LFSR can also be used for a data-compression method. Here, test response data are taken from the device-under-test and entered serially into a single-input signature register (SISR), or in parallel into a multiple-input signature register (MISR). If the LFSR is k bits long, the test response is compressed into a data word (signature) of k bits stored in the LFSR at the end of the test sequence. A faulty circuit would output a different sequence to the SISR/MISR, causing the signature to differ from the good machine signature. All single-bit errors can be detected by an LFSR with at least two feedback taps (at least two of its flip-flops feeding XOR gates in the feedback network). There are aliasing problems, however, occurring when a faulty test response gets compressed by the LFSR and has the same bit pattern as the correct signature. The probability of not detecting an error for large bit streams is based on the length of the LFSR. If all bit positions in the test response are equally likely to be in error, the probability of an error being undetected is equal to 2^{-k}. So, by increasing the length of the signature analyzer, the aliasing probability is reduced, but it will always be nonzero. For example,

if a 17-bit MISR is used, its aliasing probability is 7.63×10^{-6}. Figure 18.5(b) illustrates a SISR, and Figure 18.5(c) a MISR.

In addition to the structural self-test, there are another two hardware-based self-test techniques: the ad-hoc test and the scan-based test. In general, these BIST techniques can be categorized into two variants. The first is memory-BIST, which is aimed at testing memories, and the second is logic-BIST, which is aimed at testing logic blocks.

(a) Memory-BIST

This class of BIST technique is composed of controller logic which uses various algorithms to generate input patterns that are used to exercise the memory elements of a test design (say a RAM). The logic is automatically generated, based on the size and configuration of the memory element, generally in the form of synthesizable Verilog or VHDL (very high-speed integrated-circuit), which is inserted in the register-transfer-level source with hookups, leading to the memory elements. Upon triggering, the BIST logic generates input patterns based on a predefined algorithm, to fully examine the memory elements. The output result is fed back to the BIST logic, where a comparator is used to compare what went in against what was read out. The output of the comparator generates a pass/fail signal that signifies the authenticity of the memory elements.

(b) Logic-BIST

A logic-BIST technique uses the same approach, but targets the logic part of the design. It uses a random pattern generator to exercise the scan chains in the design. The output is a compressed signature that is compared against a simulated signature. If the two match, the device passes; otherwise it fails. The main advantage of this approach is that it eliminates the need for test engineers to generate huge scan vectors as inputs to the device under test, which saves a tremendous amount of test time. The disadvantage is that additional logic (thus area) must be incorporated within the design, just for testing purposes.

(3) Software-based embedded processor self-test

Figure 18.6 depicts the basic concept of embedded processor testing by executing software-based self-test routines. Unlike a hardware-based self-test, this type is non-intrusive; it applies tests in the circuit's normal operational mode. Moreover, software instructions can guide the test patterns through a complex processor, avoiding test data blockage arising from nonfunctional control signals, as occurs with hardware-based logic-BIST.

As shown in Figure 18.6, the self-test application program is downloaded into the embedded instruction memory of the processor via external test equipment which has access to the processor bus. The self-test data are then downloaded to embedded data memory. Control is transferred to self-test program, which starts the test execution. During this stage, test patterns are applied to internal processor components via processor instructions to detect their faults. Components' responses are collected in registers and/or data memory locations, either in an unrolled manner, or compacted, by using any known test response compaction algorithm. After the self-test code completes execution, the test responses are transferred to the external tester for evaluation.

The test data in Figure 18.6 may consist of, among other items: (1) parameters, variables and constants of the embedded code, (2) test patterns that will be explicitly applied to internal processor modules for their testing, and (3) the expected fault-free test responses to be compared

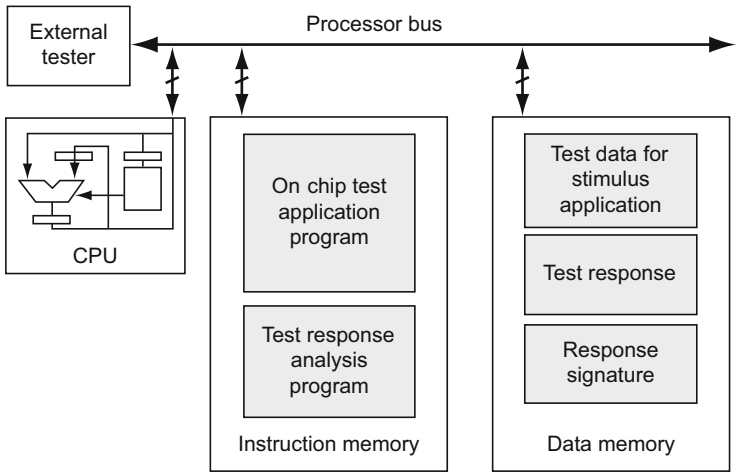


FIGURE 18.6

Embedded software-based self-testing concept.

with actual test responses. The test program may be built-in in a ROM or flash memory. In this case there is no need for a downloading process and the self-test program can be used many times.

These self-test methods include two steps: test preparation and self-testing. Test preparation involves generating what we call realizable tests for the processor's components, which can be delivered through instructions. The processor executes a special software program and stores the results in memory. Analyzing the results indicates the presence of any defects. To avoid producing undeliverable test patterns, it can generate the tests under the constraints of the processor instruction set. The tests can then be either stored or generated on chip, depending on which is more efficient.

Up to now, many software-based self-test methods for the embedded processors in SoC architecture have been developed. Two of them are widely used in industry.

(a) Stuck-at fault testing

A fault model is an engineering model of something that could go wrong in the construction or operation of a piece of equipment. Such models can be used to predict the consequences of a given fault. A stuck-at fault is a particular model used by simulators and automatic test pattern generation tools to mimic a manufacturing defect within an integrated circuit. Individual signals and pins are assumed to be stuck at logical "1", "0" and "X". For example, an output is tied to a logical "1" state during test generation to ensure that a manufacturing defect with that type of behavior can be found with a specific test pattern. Likewise the output could be tied to a logical "0" to model the behavior of a defective circuit that cannot switch its output pin.

(b) Delay fault testing

The basic idea is to test a set of interconnects, or paths, between two logic blocks for delay faults, by creating a race condition between the signals on those paths. The particular set of paths under

test are configured such that their fault-free propagation delays are nearly identical, and thus a signal transition that occurs simultaneously at the start of the paths should also occur simultaneously at their end. The bus-oriented wiring structure easily facilitates the configuration of such equal-delay paths. Any signal transition that occurs at the destination logic block substantially later than the first occurring signal transition signifies the presence of a delay fault. Note that terms such as equal, identical, simultaneous, and at the same time are not used in the pedantically correct manner, since it is impossible to have a set of paths under test with exactly the same propagation delays.

18.2 INSTALL AND CONFIGURE ROUTINES

An embedded control system must have the following types of component: (1) processor or central processing units (CPUs); (2) timer or clock units; (3) system memory units; and (4) I/O interface units to connect external devices, etc. Sometimes, other coprocessor chips may be attached, such as communications or multimedia coprocessors, to operate in parallel and execute operations that the main processor does not support. Similarly, a distributed control system consists of one or more controller devices or components to dominate the end equipments.

The install and configure routines in an industrial control system are firmware and software packages designed to understand the hardware of the system. In this section, we consider device install and device configure routines for industrial control systems. The install routine is used to work out what devices are hosted and where they are located. The configure routine is used to work out the details of each installed device including the attributes and parameters useful to implement control functionality. Without these two types of routine, application software cannot understand and record the controller or component devices, and therefore is unable to use the system's controller.

18.2.1 Bus-based embedded system install and configure

Functionally, the tasks of a bus in a bus-based embedded control system are carried out by means of two broad classes: bus master and bus slave modules. The former control the bus and initiate data transfers by driving the address and control lines. It is equipped with either a CPU or similar logic to acheive this. The master informs all other modules in the system of the type of bus cycle it starts and qualifies a valid address on the address bus by issuing proper status signals or function code signals over the control lines.

In contrast, a bus slave module cannot start bus cycles; it only monitors activity and, if addressed during a particular bus cycle, it either accepts data from the data bus or places data on it. A slave always receives and decodes address and control signals in order to determine whether or not it should respond to the current bus cycle. All data transfer activities between master and slave are implemented with "bus cycles", which will be explained in this subsection.

(1) Hierarchy of buses in an embedded system

A hierarchy of buses is usually used to interconnect the various components of an embedded control system. Some introductory comments on the types of buses are included here.

(1) Processor bus and local bus

In an embedded control system, some devices are closely coupled to the CPU by means of the processor or CPU bus. This is specified by the I/O pins in the bus interface of the CPU (Figure 5.1), and can be termed a first-level or level-1 bus. The other components of the system are interconnected through a second-level or level-2 bus, also referred to as the local bus. Often, each of these buses is functionally made up of the data bus, the address bus, and the control bus.

(2) Global or system bus

This third-level, or level-3, bus connects together all processor and ASIC (application-specific integrated-circuits) boards of the embedded control system, and is called the global or system bus. Figure 5.2 partly shows a processor board that can be connected to global (system) memory and global (system) I/O interface boards via a system bus. Although many industrial buses (such as the Multibus, VMEbus, and Futurebus, etc.) can be used as the system bus, the PCI (peripheral component interconnect) bus is the most commonly used global or system bus in embedded control systems.

(3) I/O bus

Finally, a number of peripheral devices may be connected by using special I/O buses, such as a SCSI (small computer system interface) bus used by hard drives, a PCI bus used for connecting interface devices and instruments, and LAN (local area network) interconnections. Each of these will require its respective (additional) I/O controller or bus adapter board closely connected to the global or system bus.

(2) PCI bus working mechanism

Peripheral Component Interconnect (PCI), as its name implies, is a standard that describes how to connect the peripheral components of an embedded control system together in a structured and controlled way. This standard describes the way that system components should be electrically connected and the way that they should behave.

Figure 18.7 is the logical diagram of an assumed PCI-based embedded system. The PCI buses and PCI-PCI bridges are the medium that connects the system components together; the CPU is connected to PCI bus 0, as is the GUI device. A special PCI device, a PCI-PCI bridge, connects the primary bus to the secondary PCI bus, PCI bus 1. In the jargon of the PCI specification, PCI bus 1 is described as being downstream of the PCI-PCI bridge and PCI bus 0 is upstream of the bridge. Connected to the secondary PCI bus are the SCSI and Ethernet devices for the system. Physically this bridge, the secondary PCI bus, and two devices would all be contained on the same combination PCI card. The PCI-ISA bridge in the system supports older, legacy ISA (industry standard architecture, that was a computer bus standard for IBM compatible computers) devices and this figure shows a super I/O controller chip, which could be controlling the floppy disk or the keyboard.

(1) PCI address spaces

The CPU and all the PCI devices need to access memory that is shared by them. Device drivers control the PCI devices and pass information between them by using this memory. Typically, this shared memory contains control and status registers for the device, which are used to control the device and to read its status. For example, the PCI SCSI device driver would read its status register to find out

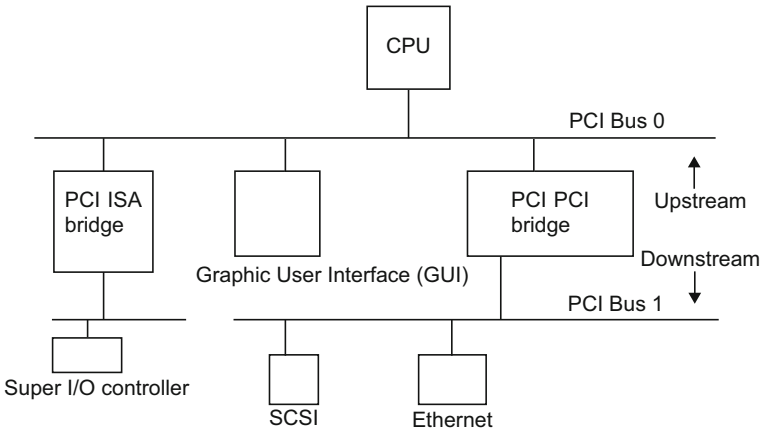


FIGURE 18.7

A PCI-based embedded system.

whether the device was ready to write a block of information, or it might write to the control register to start the device after it has been turned on.

The CPU's system memory could be used for this shared memory but in this case, every time a PCI device accessed memory, the CPU would have to stall, waiting for it to finish. Access to memory is generally limited to one system component at a time. This would slow the system down. It does not allow the system's peripheral devices to access main memory in an uncontrolled way. This would be very dangerous; a malfunctioning device could make the system very unstable.

Peripheral devices have their own memory spaces. The CPU can access these spaces, but access by the devices to the system's memory is very strictly controlled, by using DMA (direct memory access) channels. ISA devices have access to two address spaces; ISA I/O (input/output) and ISA memory. With most advanced microprocessors, PCI must have three elements: PCI I/O, PCI memory, and PCI configuration space.

Some microprocessors, for example, the Alpha AXP processor, do not have natural access to address spaces other than the system address space. This processor uses support chipsets to access other address spaces such as the PCI configuration space by use of a sparse address-mapping scheme that steals part of the large virtual address space and maps it to the PCI address spaces.

(2) PCI configuration headers

Every PCI device in an embedded system, including the PCI-PCI bridge, has a configuration data structure that is stored somewhere in the PCI configuration or address space. The configuration header of this structure allows the system to identify and to control the device. Exactly where the header is in the PCI configuration or address space depends on where in the PCI topology that device is. For example, a GUI card plugged into one PCI slot on the PC motherboard will have its configuration header at one location but if it is plugged into another PCI slot then its header will appear in another location in PCI configuration memory. This does not matter, for wherever the PCI devices and bridges

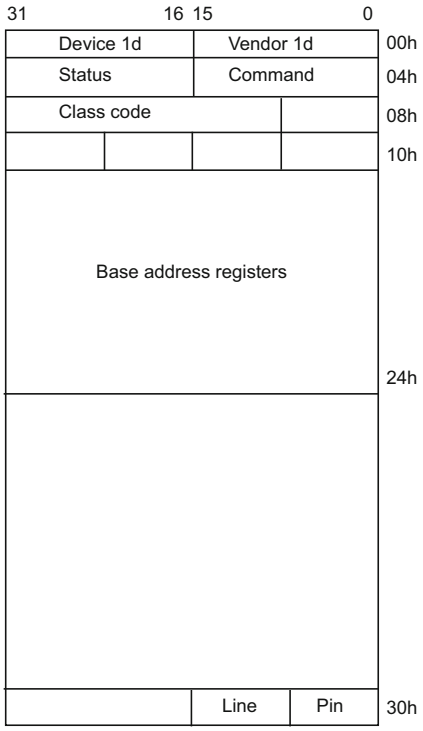


FIGURE 18.8

The PCI configuration header.

are, the system will find and configure them, using the status and configuration registers in their configuration headers.

Typically, systems are designed so that every PCI slot has its own configuration header in an offset that is related to its slot on the board. So, for example, the first slot on the board might have its PCI configuration at offset 0 and the second slot at offset 256 (all headers are the same length; 256 bytes) and so on.

A system-specific hardware mechanism is defined so that the PCI configuration code can attempt to examine all possible configuration headers for a given PCI bus and know which devices are present simply by trying to read one of the fields in the header (usually the Vendor Identification field) and getting some sort of error. This describes one possible error message as returning 0xFFFFFFFF when attempting to read the Vendor Identification and Device Identification fields for an empty PCI slot.

Figure 18.8 shows the layout of the 256 bytes of PCI configuration header. It contains the fields as are listed in Table 18.1.

(3) PCI I/O and PCI memory addresses

Devices communicate with drivers running in the kernel via these two address spaces. For example, the DEC chipset 21141 fast Ethernet device maps its internal registers into PCI I/O space. Its device driver

Table 18.1 The Fields of the 256-Byte PCI Configuration Header

Vendor Identification	A unique number describing the originator of the PCI device. Digital's PCI Vendor Identification is 0x1011 and Intel's is 0x8086
Device Identification	A unique number describing the device itself. For example, Digital's 21141 fast Ethernet device has a device identification of 0x0009
Status	This field gives the status of the device with the meaning of the bits of this field set by the standard
Command	By writing to this field the system controls the device, for example, allowing the device to access PCI I/O memory
Class Code	This identifies the type of device that this is. There are standard classes for every sort of device: GUI, SCSI, and so on. The class code for SCSI is 0x0100
Base Address Registers	These registers are used to determine and allocate the type, amount, and location of PCI I/O and PCI memory space that the device can use
Interrupt Pin	Four of the physical pins on the PCI card carry interrupts from the card to the PCI bus. The standard labels these as A, B, C, and D. The Interrupt Pin field describes which of these pins this PCI device uses. Generally it is hardwired for a particular device. That is, every time the system boots, the device uses the same interrupt pin. This information allows the interrupt handling subsystem to manage interrupts from this device
Interrupt Line	The Interrupt Line field of the device's PCI configuration header is used to pass an interrupt handle between the PCI initialization code, the device's driver, and the operating system's interrupt handling subsystem. The number written there is meaningless to the device driver but it allows the interrupt handler to correctly route an interrupt from the PCI device to the correct device driver's interrupt handling code within the operating system

then reads and writes those registers to control the device. GUI drivers typically use large amounts of PCI memory space. Until the PCI system has been set up and the device's access to these address spaces has been turned on using the Command field in the PCI configuration header, nothing can access them. It should be noted that only PCI configuration code reads or writes PCI configuration addresses; the device drivers only read and write PCI I/O and PCI memory addresses (this is again left to the device driver's policies).

(4) PCI-ISA bridges

These bridges support legacy ISA devices by translating PCI I/O and PCI memory space accesses into ISA I/O and ISA memory accesses. A lot of systems now sold contain both types of slot. Over time the need for this backwards compatibility will dwindle and PCI-only systems will be sold. Where in the ISA address spaces (I/O and memory) the ISA devices of the system have their registers was fixed in the dim mists of time by the early Intel 8080-based PCs. The PCI specification copes with this by reserving the lower regions of the PCI I/O and PCI memory address spaces for use by ISA peripherals in the system, and by using a single PCI-ISA bridge to translate PCI accesses into ISA accesses.

(5) PCI-PCI bridges

PCI-PCI bridges are special PCI devices that glue the PCI buses of the system together. Simple systems have a single PCI bus but there is an electrical limit on the number of devices that a single bus can

support. Using PCI-PCI bridges to add more buses allows the system to support many more PCI devices. This is particularly important for a high-performance server.

(i) PCI-PCI bridges: PCI I/O and PCI memory windows. PCI-PCI bridges only pass a subset of I/O and memory read and write requests downstream. For example, in Figure 18.7 the PCI-PCI bridge will only pass the reading and written addresses from PCI bus 0 to PCI bus 1 if they are for I/O or memory addresses owned by either the SCSI or Ethernet device; all others are ignored. This stops addresses propagating needlessly throughout the system. To do this, the PCI-PCI bridges must be programmed with a base and limit for I/O and memory space access that they have to pass from their primary bus onto their secondary bus. Once the bridges have been configured, then so long as the device drivers only access PCI I/O and PCI memory space through these windows, they are invisible. This is an important feature that makes life easier for Linux PCI device driver writers, for instance.

(ii) PCI-PCI bridges: configuration cycles and bus numbering. So that the CPU's PCI initialization code can address devices that are not on the main PCI bus, there has to be a mechanism that allows bridges to decide whether or not to pass configuration cycles from their primary to their secondary interface. A cycle is just an address as it appears on the bus. The PCI specification defines two formats for the PCI configuration addresses, Type 0 and Type 1; these are shown in Figure 18.9(a) and (b), respectively.

Type 0 PCI configuration cycles do not contain a bus number and these are interpreted by all devices as being for PCI configuration addresses on this PCI bus. Bits 31:11 of the Type 0 configuration cycles are treated as the device select field. One way to design a system is to have each bit select a different device. In this case, bit 11 would select the PCI device in slot 0, bit 12 would select the PCI device in slot 1, and so on. Another way is to write the device's slot number directly into bits 31:11. Which mechanism is used depends on the system's PCI memory controller.

Type 1 PCI configuration cycles contain a PCI bus number and all PCI devices except the PCI-PCI bridges ignore it. All of the PCI-PCI bridges seeing Type 1 configuration cycles may choose to pass them to the PCI buses downstream of themselves. Whether the bridge ignores the cycle or passes it downstream depends on how it has been configured. Every PCI-PCI bridge has a primary bus interface number and a secondary bus interface number. The primary interface is the one nearest the CPU, and the secondary one is furthest away. Each bridge also has a subordinate bus number, which is the maximum bus number of all the PCI buses that are bridged beyond the secondary bus interface. To put

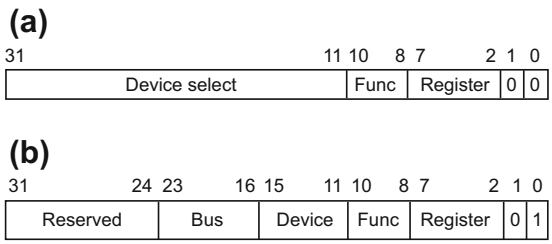


FIGURE 18.9

Two types of PCI configuration cycle: (a) Type 0; (b) Type 1.

it another way, the subordinate bus number is the highest numbered PCI bus downstream of the PCI-PCI bridge. When it sees a Type 1 PCI configuration cycle it does one of the following things:

(a) ignores it if the bus number specified is not between the bridge's secondary bus number and subordinate bus number (inclusive);
(b) converts it to a Type 0 configuration command if the bus number specified matches the secondary bus number of the bridge;
(c) passes it on to the secondary bus interface unchanged if the bus number specified is greater than the secondary bus number and less than or equal to the subordinate bus number.

So, if we want to address Device 1 on Bus 3 of the topology we must generate a Type 1 configuration command from the CPU. Bridge 1 passes this unchanged onto Bus 1. Bridge 2 ignores it, but Bridge 3 converts it into a Type 0 configuration command and sends it out on Bus 3 where Device 1 responds to it.

(6) PCI initialization

In PCI-based embedded system, the initialization code can be broken into three logical parts:
(i) PCI device driver. This pseudo-device driver searches the PCI system starting at Bus 0 and locates all PCI devices and bridges. It builds a linked list of data structures describing the topology of the system and numbers all of the bridges that it finds.
(ii) PCI BIOS. This is the software layer that should provide the various services required for PCI. This is implemented differently by every operating system.
(iii) PCI firmware. System-specific firmware code tidies up the loose ends of PCI initialization.

(7) The PCI device driver

The PCI device driver is not really a device driver at all, but a function of the operating system called at system initialization time. The PCI initialization code must scan all of the PCI buses in the system looking for PCI devices (including PCI-PCI bridge devices).

It uses the PCI BIOS code to find out if every possible slot in the current bus is occupied. If the PCI slot is occupied, it builds a PCI DEV data structure describing the device and links into the list of known PCI devices.

The PCI initialization code starts by scanning PCI Bus 0. It tries to read the Vendor Identification and Device Identification fields for every possible PCI device in every possible PCI slot. When it finds an occupied slot it builds a PCI DEV data structure describing the device. All of the PCI DEV data structures built by the PCI initialization code (including all of the PCI-PCI bridges) are linked into a singly linked list: PCI DEV.
(i) Configuring PCI-PCI bridges: assigning PCI bus numbers. For PCI-PCI bridges to pass I/O, memory, or configuration address space reads and writes across them, they need to know the following:

(a) Primary bus number. The bus number immediately upstream of the PCI-PCI bridge.
(b) Secondary bus number. The bus number immediately downstream of the PCI-PCI bridge.
(c) Subordinate bus number. The highest bus number of all of the buses that can be reached downstream of the bridge.
(d) PCI I/O and PCI memory windows. The window base and size for I/O address space and memory address space for all addresses downstream of the PCI-PCI bridge.

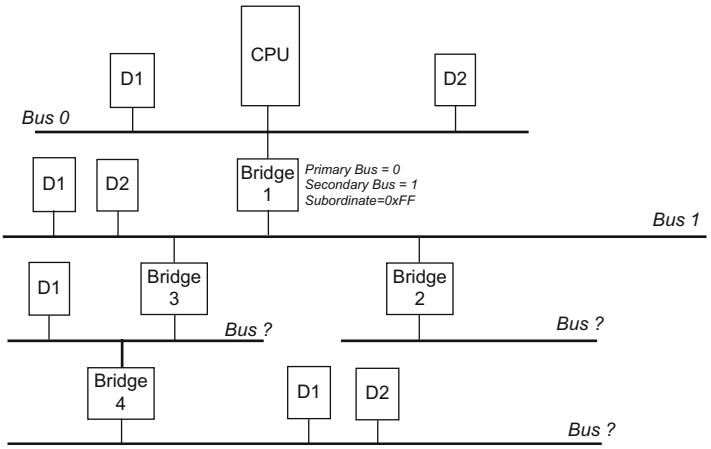


FIGURE 18.10

Configuring a PCI-based embedded system: step 1.

The problem is that at the time when you wish to configure any given PCI-PCI bridge you do not know the subordinate bus number for that bridge. You do not know whether there are further PCI-PCI bridges downstream and if you did, you would not know what numbers will be assigned to them. The answer is to use a depth recursive algorithm and scan each bus for any PCI-PCI bridges, assigning them the numbers as they are found. As each PCI-PCI bridge is found and its secondary bus numbered, assign it a temporary subordinate number of 0xFF and scan and assign numbers to all PCI-PCI bridges downstream of it. This all seems complicated but the example below makes this process clearer.

(ii) PCI-PCI bridge numbering: step 1 (the Linux approach). Taking the topology in Figure 18.10, the first bridge the scan would find is Bridge 1. The downstream PCI bus would be numbered as 1 and Bridge 1 assigned a secondary bus number of 1 and a temporary subordinate bus number of 0xFF. This means that all Type 1 PCI configuration addresses specifying a PCI bus number of 1 or higher would be passed across Bridge 1 and onto PCI Bus 1. They would be translated into Type 0 configuration cycles if they have a bus number of 1 but left unable to be translated for all other bus numbers. This is exactly what the PCI initialization code needs to do to go and scan PCI Bus 1.

(iii) PCI-PCI bridge numbering: step 2 (the Linux approach). Linux uses a depth algorithm and so the initialization code goes on to scan PCI Bus 1. Here it finds PCI-PCI Bridge 2. There are no further PCI-PCI bridges beyond PCI-PCI Bridge 2, so it is assigned a subordinate bus number of 2 that matches the number assigned to its secondary interface. Figure 18.11 shows how the buses and PCI-PCI bridges are numbered at this point.

(iv) PCI-PCI bridge numbering: step 3 (the Linux approach). The PCI initialization code returns to scanning PCI Bus 1 and finds another PCI-PCI bridge, Bridge 3. It is assigned 1 as its primary bus interface number, 3 as its secondary bus interface number, and 0xFF as its subordinate bus number. Figure 18.12 shows how the system is configured now. Type 1 PCI configuration cycles with a bus number of 1, 2, or 3 will be correctly delivered to the appropriate PCI buses.

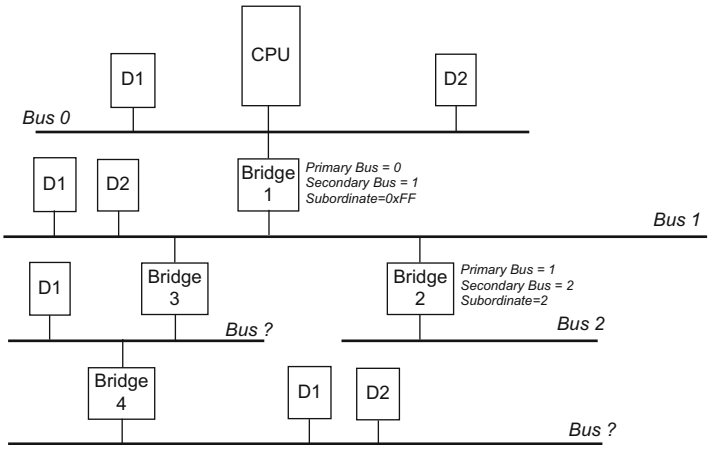


FIGURE 18.11

Configuring a PCI-based embedded system: step 2.

(v) PCI-PCI bridge numbering: step 4 (the Linux approach). Linux starts scanning PCI Bus 3, downstream of PCI-PCI Bridge 3. PCI Bus 3 has another PCI-PCI bridge (Bridge 4) on it. It is assigned 3 as its primary bus number and 4 as its secondary bus number. It is the last bridge on this branch and so it is assigned a subordinate bus interface number of 4. The initialization code returns to PCI-PCI Bridge 3 and assigns it a subordinate bus number of 4. Finally, the PCI initialization code can assign 4 as the subordinate bus number for PCI-PCI Bridge 1. Figure 18.13 shows the final bus numbers.

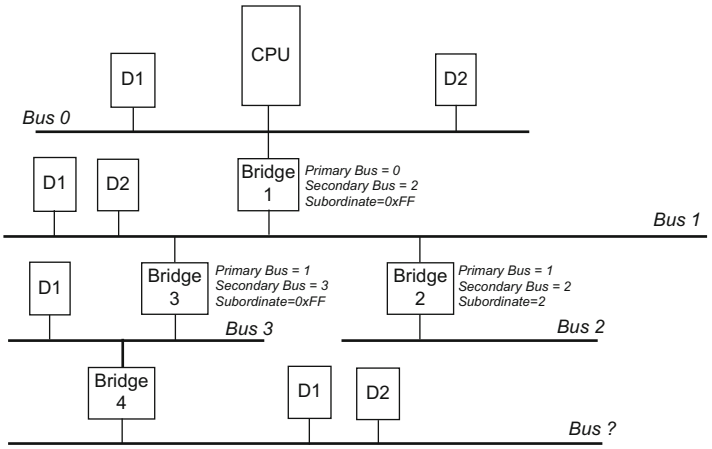


FIGURE 18.12

Configuring a PCI-based embedded system: step 3.

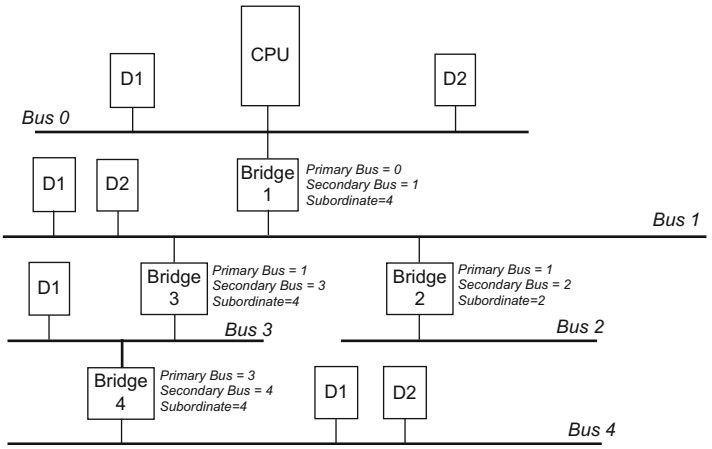


FIGURE 18.13

Configuring a PCI-based embedded system: step 4.

(8) PCI BIOS functions

These functions are a series of standard routines that are common across all platforms for example, they are the same for both Intel and Alpha AXP-based systems. They allow CPU controlled access to all of the PCI address spaces. Therefore, only Linux kernel code and device drivers may use them.

(9) PCI firmware

The PCI firmware code for Alpha AXP does rather more than that for Intel (which basically does nothing). For Intel-based systems the system BIOS, which ran at boot time, has already fully configured the PCI system. For non-Intel-based systems further configuration needs to happen:

(i) allocate PCI I/O and PCI memory space to each device;
(ii) configure the PCI I/O and PCI memory address windows for each PCI-PCI bridge in the system;
(iii) generate Interrupt Line values for the devices; these control the interrupt handling for the device.

The following paragraphs describe how that the PCI BIOS and firmware code work for installing and configuring.

(1) Finding out how much PCI I/O and PCI memory space a device needs. Each PCI device found is queried to find out how much PCI I/O and PCI memory address space it requires. To do this, each base address register has all 1s written to it and then read. The device will return 0s in the do-not-care address bits, effectively specifying the address space required.

There are two basic types of base address register; the first indicates within which address space the device registers must reside, either PCI I/O or PCI memory space. This is indicated by Bit 0 of the register. Figure 18.14 shows the two forms of the base address register for PCI memory and for PCI I/O. To find out just how much of each address space a given base address register is requesting, you write all 1s into the register and then read it back. The device will specify zeros in those do-not-care address bits, effectively specifying the address space required. This design implies that all address spaces used are a power of 2 and are naturally aligned.

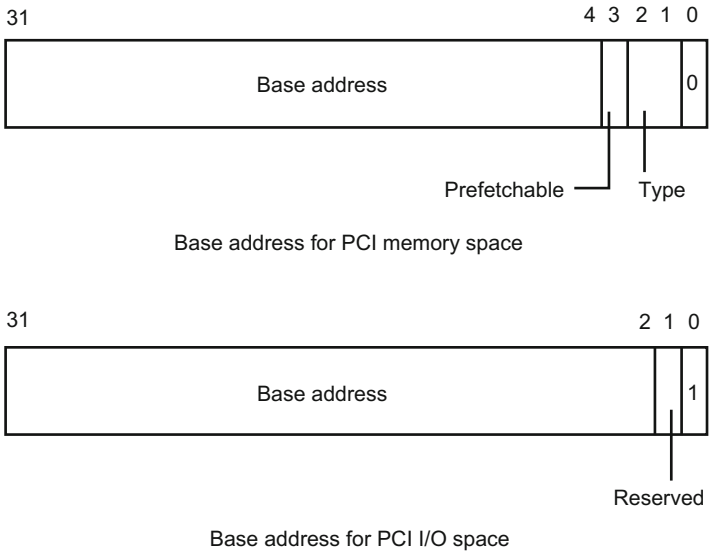


FIGURE 18.14

PCI configuration header: base address registers.

(2) Allocating PCI I/O and PCI memory to PCI-PCI bridges and devices. Similar to all memory, the PCI I/O and PCI memory spaces are finite, and to some extent scarce. The PCI firmware code for non-Intel systems (and the BIOS code for Intel systems) has to allocate each device the amount of memory that it is requesting in an efficient manner. Both PCI I/O and PCI memory must be allocated to a device in a naturally aligned way. For example, if a device asks for 0xB0 of PCI I/O space, then it must be aligned on an address that is a multiple of 0xB0. In addition to this, the PCI I/O and PCI memory bases for any given bridge must be aligned on 4 k and on 1 MB boundaries, respectively. Given that the address spaces for downstream devices must lie within all of the upstream PCI-PCI bridge's memory ranges for any given device, it is a somewhat difficult problem to allocate space efficiently.

A recursive algorithm can be used to walk through the data structures built by the PCI initialization code. Starting at the root PCI bus the BIOS firmware code:

(a) aligns the current global PCI I/O and memory bases on 4 k and 1 MB boundaries, respectively;
(b) for every device on the current bus (in ascending PCI I/O memory needs): it allocates its space in PCI I/O and/or PCI memory; then, it moves on the global PCI I/O and memory bases by the appropriate amounts; enables the device's use of PCI I/O and PCI memory;
(c) allocates space recursively to all of the buses downstream of the current bus; note that this will change the global PCI I/O and memory bases;
(d) aligns the current global PCI I/O and memory bases on 4 k and 1 MB boundaries, respectively, and in doing so figures out the size and base of PCI I/O and PCI memory windows required by the current PCI-PCI bridge;
(e) programs the PCI-PCI bridge that links to this bus with its PCI I/O and PCI memory bases and limits;

(f) turns on bridging of PCI I/O and PCI memory accesses in the PCI-PCI bridge; this means that any PCI I/O or PCI memory addresses seen on the bridge's primary PCI bus that are within its PCI I/O and PCI memory address windows will be bridged onto its secondary PCI bus.

Taking the PCI system in Fig. 18.7 as our example, the PCI firmware code would set up the system in the following way:

1. Align the PCI bases. PCI I/O is 0x4000 and PCI memory is 0x100000. This allows the PCI-ISA bridges to translate all addresses below these into ISA address cycles,
2. The GUI device. This is asking for 0x200000 of PCI memory and so we allocate it that amount starting at the current PCI memory base of 0x200000 as it has to be naturally aligned to the size requested. The PCI memory base is moved to 0x400000 and the PCI I/O base remains at 0x4000.
3. The PCI-PCI bridge. We now cross the PCI-PCI bridge and allocate PCI memory there; note that we do not need to align the bases as they are already correctly aligned. (a) The Ethernet device. This is asking for 0xB0 bytes of both PCI I/O and PCI memory space. It gets allocated PCI I/O at 0x4000 and PCI memory at 0x400000. The PCI memory base is moved to 0x4000B0 and the PCI I/O base to 0x40B0. (b) The SCSI device. This is asking for 0x1000 PCI memory and so it is allocated it at 0x401000 after it has been naturally aligned. The PCI I/O base is still 0x40B0 and the PCI memory base has been moved to 0x402000.
4. The PCI-PCI bridge's PCI I/O and memory windows. We now return to the bridge and set its PCI I/O window at between 0x4000 and 0x40B0 and its PCI memory window at between 0x400000 and 0x402000. This means that the PCI-PCI bridge will ignore the PCI memory accesses for the GUI device and pass them on if they are for the Ethernet or SCSI devices.

18.2.2 Device install and configure routines

For an industrial control system, the device install and configure routine is resident in the respective microprocessor-unit. This means that each board or chipset must have its device install and configure routine to handle those components and devices that it controls directly. However, even in the same system, the device install and configure routine for one microprocessor-unit board or chipset may be different from the routines for other units.

The routine takes the following responsibilities: (1) detecting the existence of each device and retaining the results; (2) allocating system bus I/O and/or memory spaces to each existing device for storing the configuration data; (3) initializing each existing device's configuration data; and (4) initializing the driver in operating system code for each existing device.

The device install and configure routine for a microprocessor-unit board or chipset is basically realized with its electronic hardware. For an industrial control system working with the PCI mechanism, the PCI initialization code of this board dominates the installation and configuration of the devices resident on a microprocessor-unit board or chipset. The PCI initialization code, as elucidated in detail in subsection 18.2.1, includes these three parts: (1) PCI device driver; (2) PCI BIOS; (3) PCI firmware. Subsection 18.2.1 has introduced the procedure and the methodology for the PCI-based system's initialization code to install and configure the PCI devices, so this is not repeated.

The routine should be executed as a part of the power-on process. When this precedes system bus initialization, the routine is inclusively executed. However, as mentioned in subsection 18.2.1, whether

or not the system bus initialization can be completed during the booting depends on whether or not the microprocessor controlling the devices is an Intel processor. For such a system, the system bus BIOS fully configures the system bus during booting, but for others further configuration needs to be performed afterwards.

18.2.3 System install and configure routines

Modern industrial control systems are usually distributed systems which has some devices directly connected, and a motherboard, necessary to coordinate the whole system.

It is obvious that the device install and configure routines given in subsection 18.2.2 do not gather all the devices in an industrial control system, because each chipset has its own routine, specific to the devices connecting to that unit. We therefore need a system configure routine that understands all the devices in a distributed control system to configure all devices, and to set up and store a record of the configuration data.

This routine resides in the operating system of the motherboard. It should be emphasized that the system configure routine, unlike the device install and configure routine, is purely software.

The system configure routine is normally executed during the power-on process, but can be run at any time. It must be run after all the microprocessor units complete their own device install and configure routines during the power-on process.

This routines roughly follows these steps. (1) The motherboard starts to run this routine after it ensures all the microprocessor units of the system have done their respective device install and configure routines. (2) It sends a message to the first connected microprocessor unit to ask for the configuration data of all its devices and those of its children. (3) The motherboard repeats this step until it exhausts all the connected microprocessor units and all the system's devices.

18.3 DIAGNOSIS ROUTINES

With increasing demands for efficiency and product quality, and continuing integration of industrial control systems in high-cost mechatronic and safety-critical processes, the diagnosis and detection of faults or errors plays an important role. To achieve productive and economical usage of an industrial control system, operators must be able to rapidly recognize and eliminate causes of faults or errors to minimize operation stoppages. This is only possible through the use of powerful diagnostic methods, equipment, and routines. A differentiation is made between two types of diagnostics:

1. Process diagnostics: this means the detection and elimination of faults in production such as manufacturing and chemical processes, i.e. falling outside their control system. These faults could result from deficiencies in the production process such as a wrong value of a control parameter or some bugs in the processing or controlling programs.
2. System diagnostics: this means the localizing and the elimination of faults in the industrial control system. Control system faults are hardware component faults generated in its control level (also called master level), its interface level, or in its slave level. System diagnostics is of particular significance because it is always required.

In this section, we first discuss the basic diagnosis and detection methods of faults and errors, and then introduce some important diagnosis equipments and routines.

18.3.1 Fault diagnosis methods

A running process or a unit under test fails when its observed behavior is different from that expected. Diagnosis consists of locating the physical fault(s) in a structural model of the process. The degree of accuracy to which faults can be located is called diagnostic resolution. Functionally equivalent faults cannot be distinguished. The partition of all faults into distinct subsets of functionally equivalent faults defines the maximal fault resolution. A test that achieves the maximal fault resolution is said to be a complete fault-location test.

Repairing the running process or unit often consists of substituting one of its replaceable units, referred to as a faulty replaceable unit, rather than in an accurate identification of the real fault inside a unit. This is characterized by us as replaceable unit resolution. Suppose that the results of the test cannot distinguish between two suspected replaceable units U1 and U2. We could replace one of these replaceable units, say U1, and return to the test experiment. If the new results are correct, the faulty replaceable unit was the replaced one; otherwise, it is the remaining one U2. This type of procedure we call the sequential diagnosis procedure.

The diagnosis process is often hierarchical, carried out in a top-down fashion (with a system operating in the field) or as a bottom-up process (during the fabrication of the system).

1. The top-down approach is carried out from system to units to boards and finally to integrated circuits. The first-level diagnosis may deal with "large" replaceable units such as boards, also called field-replaceable units. The faulty board is then tested in a maintenance center to locate the faulty integrated circuit on it. Accurate location of faults inside a faulty integrated circuit may be also useful for improving the manufacturing process.
2. The bottom-up approach is carried out from integrated circuits to boards to units and finally to the system. In this approach, a higher level is assembled only from integrated circuits already tested at a lower level. This is done to minimize the cost of diagnosis and repair, which increases with the level at which the faults are detected.

In industrial control systems such as SCADA and distributed systems, the most likely faults are fabrication errors affecting the interconnections between units or subsystems in the field. In industrial processes such as manufacturing and chemical processes, the most likely faults are physical failures internal to units or even to integrated circuits (because every process or system under test or operation has been successfully tested in the past). Therefore, knowing the most likely class of faults helps us in fault location.

(1) Combinational fault diagnosis method

This method does most of the work before the testing experiment. It uses fault simulation to determine the possible responses to a given test in the presence of faults. The database constructed in this step is called a fault table or a fault dictionary. To locate faults, one tries to match the actual results of test experiments with one of the expected results stored in the database. The result of the test experiment represents a combination of effects of the fault on each test pattern. That is why this approach is defined as the combinational fault diagnosis

method. If this look-up process is successful, the fault table (dictionary) indicates the corresponding fault(s).

(1) Fault database: fault table and fault dictionary

In general, a fault table is a matrix FT = $\{a_{ij}\}$ where columns F_j represent faults, rows T_i represent test patterns, and $a_{ij} = 1$ if the test pattern T_i detects the fault F_j, otherwise if the test pattern T_i does not detect the fault F_j, $a_{ij} = 0$. The result of a test experiment is represented by a vector E = $<e_i>$ where $e_i = 1$ if the actual result of the test patterns does not match with the expected result, otherwise $e_i = 0$.

Three cases are now possible, depending on the quality of the test patterns used for carrying out the test experiment:

(i) The test result E matches with a single column vector f_j in FT. This result corresponds to the case where a single fault F_j has been located. In other words, the maximum diagnostic resolution has been obtained.

(ii) The test result E matches with a subset of column vectors $<f_i, f_j \ldots f_k>$ in FT. This result corresponds to the case where a subset of indistinguishable faults $<F_i, F_j \ldots F_k>$ has been located.

(iii) No match for E with column vectors in FT is obtained. This result corresponds to the case where the given set of vectors does not allow carrying out fault diagnosis. The set of faults described in the fault table must be incomplete; in other words, the real existing fault is missing from the fault list considered in the matrix FT.

Figure 18.15 is an example in which the results of three test experiments E_1, E_2, E_3 are demonstrated. E_1 corresponds to the first case, where a single fault is located, E_2 corresponds to the second case where a subset of two indistinguishable faults is located, and E_3 corresponds to the third case where no fault can be located because of the mismatch of E_3 with the column vectors in the fault table.

A fault dictionary contains the same data as the fault tables, but the data are reorganized. Here, a mapping between the potential results of test experiments and the faults is represented in a more compressed and ordered form. For example, the column bit vectors can be represented by ordered decimal codes (see the example in Figure 18.16) or by some kind of compressed signature.

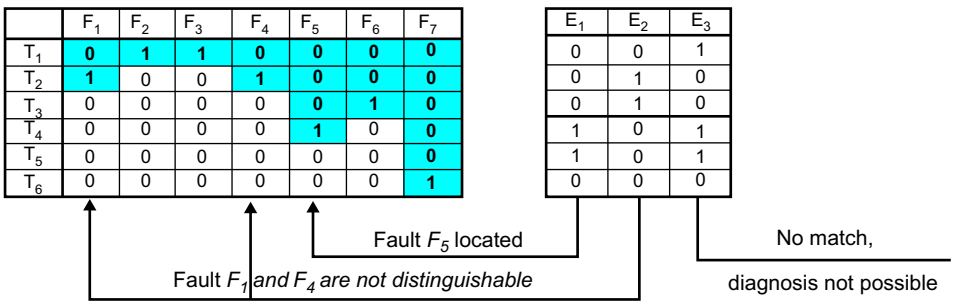


FIGURE 18.15

An example of fault table and the results of three test experiments.

No	Bit vectors	Decimal numbers	Faults
1	000001	01	F_7
2	000110	06	F_5
3	001011	11	F_6
4	011000	24	F_1, F_4
5	100011	35	F_3
6	101100	44	F_2

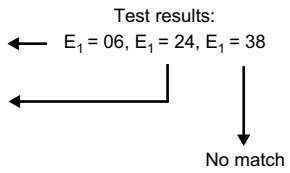


FIGURE 18.16

An example of fault dictionary which contains the results of three test experiments.

(2) Minimization of diagnostic data

To reduce large computational efforts involved in building a fault dictionary, a fault simulation can be implemented so that detected faults are dropped from the set of simulated ones. Hence, all the faults detected for the first time by the same vector will produce the same column vector (signature) in the fault table, and will be included in the same equivalence class of faults. In this case the testing experiment can stop after the first failing test, because the information provided by the following tests is not used. Such a testing experiment achieves a lower diagnostic resolution. A tradeoff between computing time and diagnostic resolution can be achieved by dropping faults in all detections after the first detection.

For example, in the fault table produced by fault simulation with fault dropping, only 19 faults need to be simulated compared to 42 faults when simulation without fault dropping is carried out (the simulated faults in the fault table are shown in shadowed boxes in Figure 18.15). As the result of the fault dropping, however, the following faults remain indistinguishable: {F2, F3},{F1, F4},{F2, F6}.

(3) Fault location by structural analysis

Assume a single fault in an integrated circuit. There should be a path from the site of the fault to each of the outputs where errors have been detected. Hence the fault site should belong to the intersection of cones of all failing outputs. A simple structural analysis of the integrated circuit can help to find faults that can explain all the observed errors.

(2) Sequential fault diagnosis method

In sequential fault diagnosis the process of fault location is carried out step by step, where each step depends on the one before it. Such an experiment is called adaptive testing. Sequential experiments can be carried out, either by observing only output responses of a process or a unit under test, or by pinpointing with a special probe selected internal control points of the process or unit under test (guided probing). This procedure can be graphically represented as a diagnostic tree.

(1) Fault location by edge-pin testing

In this approach, test patterns are applied to the process or unit under test step by step. In each step, only output signals at edge-pins of the process are observed and their values are compared to the expected ones. The next test pattern to be applied depends on the result of the previous step. The diagnostic tree of this process consists of the fault-nodes (rectangles in Figure 18.17) and test-nodes

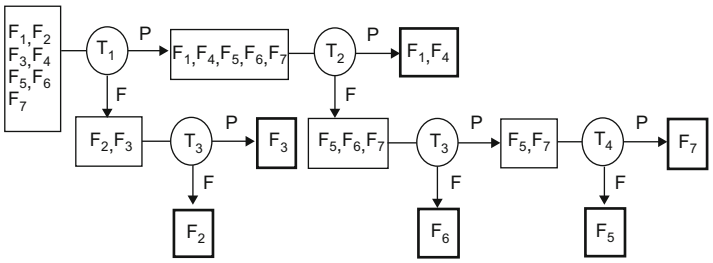


FIGURE 18.17

A diagnostic tree used for locating faults.

(circles in Figure 18.17). A fault-node is labeled by a set of not yet distinguished faults. The starting fault node is labeled by the set of all faults. To each fault-node F_k, a test-node is linked labeled by a test pattern T_k to be applied as the next experiment. Every test pattern distinguishes between the faults it detects and the ones it does not. The task of the test pattern T_k is to divide the faults in fault-node F_k into two groups: detected and not detected by T_k faults. Each test-node has two outgoing edges corresponding to the results of the experiment of this test pattern. The results are indicated as passed (P) or failed (F). The set of faults shown in a current fault-node (rectangle) are equivalent (not distinguished) under the currently applied test set.

The diagnostic tree in Figure 18.17 corresponds to the same example given in Figure 18.15 and Figure 18.16. From the former, we can see that most of the faults are uniquely identified; two F_1, F_4 remain indistinguishable. Not all test patterns used in the fault table are needed. Different faults need sequences of different lengths. The shortest test contains two patterns, and the longest four. Rather than applying the entire test sequence in a fixed order as in combinational fault diagnosis, adaptive testing determines the next vector to be applied based on the results obtained by the preceding vectors. In our example in Figure 18.17, if T_1 fails, the possible faults are $\{F_2, F_3\}$. At this point applying T_2 would be wasteful, because it does not distinguish these faults. The use of adaptive testing may substantially decrease the average number of tests required to locate a fault.

(2) Generating tests to distinguish faults

To improve the fault resolution of a given test set, it is necessary to generate tests to distinguish among the equivalent faults under the given test set. Let us consider the problem of generating a test to distinguish between faults F_1 and F_2. Please note that such a test must detect one of these faults but not the other, or vice versa. The following cases are possible: F_1 and F_2 do not influence the same set of outputs, and F_1 and F_2 influence the same set of outputs. Three possibilities can be mentioned to keep a fault F_2: $x_k = e$ not activated, where x_k denotes a line in the circuit, and e can be 0 or 1 only:

(i) The value e should be assigned to the line x_k.

(ii) If this is not possible then the activated path from F_2 should be blocked, so that the fault F_2 could not propagate and influence the activated path from F_1.

(iii) If the second case is also not possible then the values propagated from the sites F_1 and F_2 and reaching the same gate should be opposite on the inputs of the gate.

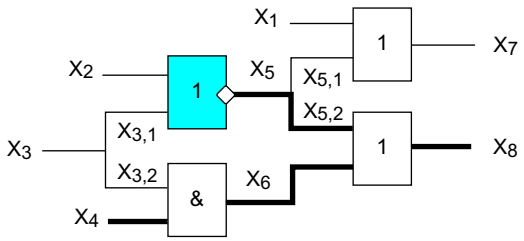


FIGURE 18.18

An example of using the generating tests to distinguish faults.

An example is illustrated in Figure 18.18 in which the following steps are followed:

(i) There are two faults in the circuit: F_1: $x_{3,1} = 0$, and F_2: $x_4 = 1$. The fault F_1 may influence both outputs, the fault F_2 may influence only the output x_8. A test pattern "0010" activates F_1 up to the both outputs, and F_2 only to x_8. If both outputs are wrong, F_1 is present, and if only the output x8 is wrong, F_2 is present.

(ii) There are two faults in the circuit: F_1: $x_{3,2} = 0$, and F_2: $x_{5,2} = 1$. Both of them influence the same output of the circuit. A test pattern "0100" activates the fault F_2. The fault F_1 is not activated, because the line $x_{3,2}$ has the same value as it would have had if F_1 were present.

(iii) There are the same two faults in the circuit: F_1: $x_{3,2} = 0$, and F_2: $x_{5,2} = 1$. Both of them influence the same output of the circuit. A test pattern "0110" activates the fault F_2. The fault F_1 is activated at its site but not propagated through the AND-gate, because of the value $x_4 = 0$ at its input.

(iv) There are two faults in the circuit: F_1: $x_{3,1} = 1$, and F_2: $x_{3,2} = 1$. A test pattern "1001" consists of the value $x_1 = 1$ which creates the condition where both of the faults may influence only the same output x_8. On the other hand, the test pattern "1001" activates both of the faults to the same OR-gate (i.e. none of them is blocked). However, the faults produce different values at the inputs of the gate, hence they are distinguished. If the output value on x_8 is 0, F_1 is present. Otherwise, if the output value on x_8 is 1, either F_2 is present; or none of the faults F_1 and F_2 is present.

(3) Guided-probe testing

Guided-probe testing extends the edge-pin testing process by monitoring internal signals in the process or unit under test via a probe, which is moved (usually by an operator) following the guidance provided by the test (or diagnosis) equipment. The principle of guided-probe testing is to backtrace an error from the primary output where it has been observed to its physical location in the process or unit under test. Probing is carried out step-by-step. At each step an internal signal is probed and compared to the expected value. The next probing depends on the result of the previous step.

In Figure 18.19, a diagnostic tree can be created for the given test pattern to control the process of probing. The tree consists of internal-nodes (circles) to mark the internal lines to be probed, and of terminal-nodes (rectangles) to show the possible result of diagnosis. The results of probing are indicated as passed (P) or failed (F).

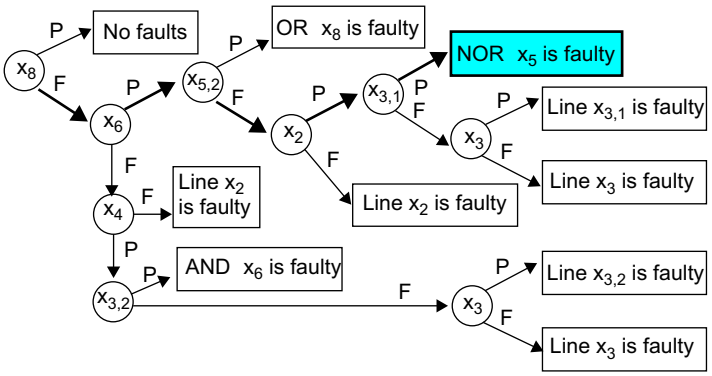


FIGURE 18.19

An example of the diagnostic tree applied for guided-probe testing.

Typical faults located are open and defective integrated-circuits. An open between two points A and B in a connection line is identified by a mismatch between the error observed at B and the correct value measured at A. A faulty device is identified by detecting an error at one of its outputs, while only correct values are measured at its inputs.

The most time-consuming part of guided-probe testing is moving the probe. To speed-up the fault location process, we need to reduce the number of probed lines. A lot of methods to minimize the number of probings are available.

As an example, let us have a test pattern "1010" applied to the inputs of the circuit. The diagnostic tree created for this particular test pattern is shown in Figure 18.19. On the output x_8, instead of the expected value 0, an erroneous signal 1 is detected. By backtracing (indicated by bold arrows in the diagnostic tree) the faulty integrated circuit NOR-x5 is located. Therefore, the diagnostic tree allows optimization of the fault location procedure, for example by generating a procedure with a minimum average number of probes.

(4) Fault location by reducing units under test

Initially the process or unit under test is the entire circuit and the test process starts when its test fails. If the failing unit can be partitioned, half of it is disabled and the remaining half is tested. If the test passes, the fault must be in the disabled part, which then becomes the new unit under test. If the test fails, the tested part becomes the new unit under test.

18.3.2 Fault diagnosis equipment

Some equipment and instruments for fault diagnosis have been developed.

(1) Fault detectors and fault recorders

Both fault detectors and fault recorders are electric power instruments that detect and/or record electrical faults that may happen in power cables. Fault detectors notify the user that a power failure

has occurred, while fault recorders keep a log of the time, location, and intensity of the fault. There are many everyday applications that require a continuous flow of electrical power to function properly, including computers, refrigerators, freezers, and hospital equipment. The results of power failure can be disastrous in certain applications, which makes it necessary to be able to locate, understand, and correct power-faults.

(2) Ground fault circuit interrupters

Ground fault circuit interrupters disconnect a circuit when the current between the neutral conductor and the live conductor becomes unbalanced. These electrical protection devices are also known as residual current devices. Ground fault circuit interrupters are available as both single devices and arrays. In a single device, there is only the ground fault circuit interrupter in the package, but in an array, there are several. When selecting a ground fault circuit interrupter array, the total number of devices in the package is an important parameter to consider.

(3) Ground fault relays

Ground fault relays protect electrical equipment from ground faults. This is an unintentional current path between a current-carrying conductor and a grounded surface. When such occurs, electric current may find a path to ground via dust, water, or worn insulation. Most short circuits in electrical equipment are caused by ground faults, which can also endanger worker safety. Some ground fault relays are designed to work with ungrounded systems. Products for motor protection may provide both predictive and protective features. Typically, output signals are sent to a remote meter or programmable logic controller (PLC).

Ground fault relays with on-line and off-line modes are designed to provide continuous protection from ground faults. Under normal conditions, these devices are used with a separately connected current transformer. Both the alarm range and the time delay are adjustable. Metered loop connections and LED (light-emitting diode) indicators are also available. If the load is switched off, an auxiliary electrical contact causes the ground fault relay to change state. The small DC (direct current) that is imposed travels through the network. Ground fault relays carry product specifications such as nominal insulation voltage, insulation ground, test voltage, supply voltage, power input, on-line current relay, and response range.

(4) Fiber-optic fault locators

Fiber-optic fault locators function by shining a red laser through jacketed fibers to identify breaks, bends, faulty connectors, splices, and other causes of signal loss. Signal loss areas will appear as bright glowing areas as a result of scattering. Fiber-optic fault locators can interface with two types of cables; single mode and multimode. Single mode is an optical fiber that will allow only one mode to propagate. The fiber has a very small core of approximately 8 μm diameter. It permits signal transmission at extremely high bandwidth and allows very long transmission distances. Multimode describes a fiber-optic cable that supports the propagation of multiple modes. It may have a typical core diameter of 50 to 100 μm with a refractive index that is graded or stepped. It allows the use of inexpensive LED light sources, and connector alignment and coupling is less critical than single mode fiber. Distances of transmission and bandwidth are less than for single-mode fiber due to dispersion. Some fiber-optic fault locators can be used for both single-mode and multimode cables.

18.3.3 Fault diagnosis routines

There are many types of fault diagnosis routines in industry. The following are some examples.

(1) Device component test routines

Industrial control systems use device component test routines, also called component control routines, to investigate the validity of device components. Validity indicates whether the conditions and statuses of the device components are adequate to perform its required function. To test this, component test routines investigate these questions. (1) Can this component communicate correctly with its master in both directions? (2) Can this component work exactly according to the commands from its master? (3) Can this component properly monitor its slave devices?

It is crucial that before starting a diagnostic routine, a system should change to a diagnostic mode in which only one session specific to the diagnostic routine is allowable. The routine can then be run without interrupts and the results are not interfered with by other tasks.

To issue the device component test routines, the motherboard's application programs establish a table containing all the components to be tested, identified by an ID code. The component ID should be defined sufficiently to discriminate one component from any other.

In general, device component test routines, once started, follow the procedure given below:

1. Change the system to a diagnostic mode.
2. Start one device component test routine.
3. Select one component from the system's component table that can be tested with this routine.
4. Send the command to the master controller of the selected component to start the test.
5. Request the test result and display the results if allowable.
6. Stop this component test.
7. Select another component that can be tested with the corresponding device component test routine, and repeat the above steps.
8. Continue untill all devices are tested.
9. Change the system mode back to normal.

(2) System NVM read and write routines

Nonvolatile memory (NVM) is a special physical medium that can maintain stored magnetic cells without electric power. It exists in almost all industrial control systems to store important parameters and working data, such as devices' physical and mechanical attributes, system installation and configuration parameters and programmed timer values, and so on. Industrial control systems use NVM to be quickly and precisely established at power-on as well as to execute software processes correctly.

Checking and modifying the NVM attributes is important for finding the root cause when software crashes. The NVM read and write routines are designed for helping the process diagnostics.

An industrial control system can have more than one NVM, each being connected through either the PCI bus or another internal bus to one microprocessor. Through the bus system, a microprocessor can communicate with the corresponding hardware controller of the respective NVM to accomplish the read or write operation.

To issue the NVM read and write routines, the application programs of the microprocessor-unit board linking to the NVM hardware controller establish a list of the NVM elements (called the NVM attributes list) to be tested. A unique ID should identify each of the NVM elements in this list.

In general, the read and write routines, once started, follow the procedure given below:

1. Change the system to a diagnostic mode.
2. Start either a NVM read or write routine.
3. Select one NVM element by its ID from the attribute list.
4. Send the command to the hardware controller of the NVM to start this read or write operation.
5. Request the result and display the results if possible.
6. Send the command to stop this read or write operation.
7. Select another NVM attribute and repeat the above steps.
8. Repeat for each element.
9. Change the system mode back to normal.

(3) Fault/error log routines

It is inevitable that an industrial control system will partially and occasionally malfunction due to (1) physical constraints of the electronics hardware, (2) material deficiencies of machinery systems, (3) code bugs in software programs, and/or (4) incorrect operation by a user or an administrator. All the causes of malfunctions are categorized as faults or errors. When one occurs, the control system may lose some services or go down. Some industrial control systems create a special system mode, called degrade mode, to represent the system state and to deal with the malfunction statuses after faults or errors are generated.

When running, a control system may generate faults at any time, and require special treatment to restore the system. Within a fixed term, the frequencies and locations of faults are important indications of system performance. The fault/error log routine is used to record frequencies and locations of faults or errors. They are useful references for analysis of the system's performance. Normally, a record of faults or errors are kept in a NVM or a disk.

If a fault or an error occurs while a control system is running, a system application program calls this routine to log the error by defined fault ID structure into a NVM or a disk as a system fault counter. Each of the logged faults and errors is identified with the defined fault ID structure, which may contain the series number, platform ID, device ID, time and positions, and so on.

The fault counter or fault record can be checked by the system NVM read routine and can be modified by the system NVM write routine. Both of these have been discussed.

One scenario of a fault/error log routine is as follows:

1. While a control system is running, an error generates;
2. System software on the basis of the impact of this error, decides whether or not to let the system enter the degraded mode, in which only some services are maintained;
3. This routine is called to log the generated error into a NVM or a special disk to be added to the system fault counter;
4. If the control system is in degraded mode, the system administrator should be informed to take some measures to handle this fault/error;
5. After the malfunction is fixed, system application programs may recover the lost services; sometimes this requires restarting the control system.

18.4 CALIBRATION ROUTINES

Calibration is a vital operation, needed in almost all industries. This section focuses on those particularly used in industrial control systems, equipment and instruments.

18.4.1 Calibration principles

The accuracy of all electronic components used in all industrial control systems drifts over time. Effects of environmental conditions increase this drift, which in turn causes errors in the functionality of the control systems. At some point, this drift causes the industrial control system to partially or totally malfunction, or even to crash. Hence, all the components and devices in a system must be calibrated at regular intervals. Typically, this involves checking the accuracy throughout the calibration ranges of the electronic component. In most cases, this range is defined by the zero and span values.

Industrial control systems require three types of calibration, as given below.

(1) System calibration

System calibration is designed to quantify and compensate for the total measurement error in industrial control systems. Cable losses, condition changes, circuit drifts, and sensor errors, etc., may induce measurement error. By applying known inputs and reviewing the resultant measurements, an error model is developed.

An error model could be as simple as a lookup table of input versus output values, or as detailed as a polynomial. Once developed, it can be applied to all measurements made with the same system.

Computer-based data acquisition and instrumentation hardware is ideal for this type of compensation because, unlike traditional box instruments, the application that defines the measurement functionality is well-developed. Thus the error compensation and control system calibration can be easily created with the application software.

(2) External calibration

When an electronic component's time in service reaches a specified calibration interval, it should be returned to the manufacturer, a suitable metrology laboratory, or metrology agency, for a calibration service. The electronic component's measurements will be compared with external standards of known accuracy. If the results of the measurements do not fall within certain specifications, adjustments are made to the measurement circuitry. In general, the act of external calibration includes the following: (1) evaluation of the component's capabilities, to determine whether it operates within specifications, (2) adjustments to measurement circuitry and onboard signal references if the component does not operate within specifications, (3) revitalization of the component to ensure that it operates within specifications, (4) issuing a calibration certificate, stating that the component measures to within specifications when compared to a traceable standard. Routine performance of external calibration ensures the accuracy of measurements made.

(3) Self-calibration

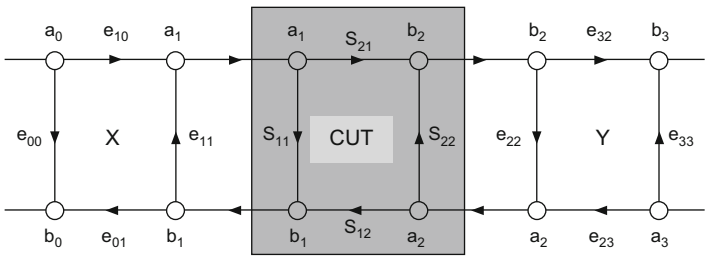
Self-calibration is a method whereby a component uses onboard signal references instead of external ones to adjust measurement accuracy. During this process, the component measures the onboard

references and adjusts its measurement capabilities. External calibration must be performed as well as self-calibration to quantify the references. A method whereby self-calibration and external calibration tools work together can be adopted to ensure the measurement accuracy of the components. Some relevant agencies' measurement products contain highly stable signal references to maintain traceability and facilitate self-calibration. Through simple software function calls, the component can attain its optimal measurement performance through self-calibration.

Before implementing an appropriate calibration, the engineer must specify an error model that is appropriate for the component under measurement. Some popular two-port models include the 8-term, 12-term and 16-term error models. Selection will depend on the working component, measurement setup and calibration frequency. Models with fewer terms are merely appropriate at lower frequencies, where the models are still valid. More complex models are required for measurements made at higher frequencies.

1. 8-term model: this is shown in Figure 18.20. The model has an error adaptor X which represents the error due to port 1 and another, Y which represents the error in port 2. The model accounts for the errors due to the finite directivity of the couplers (e_{00}, e_{33}), the port mismatch (e_{11}, e_{22}) and tracking error ($e_{01}e_{10}$, $e_{23}e_{32}$, $e_{10}e_{23}$). It assumes the leakage and cross-terms are negligible.
2. 12-term model: the 12-term model includes additional terms which account for crosstalk between the ports. The model has six error terms to model excitations in the forward direction and six terms to model excitations in the reverse direction. This model intrinsically assumes that the component under test is excited from only one test-port and thus it may not be appropriate for complex components such as distributed or network components.
3. 16-term model: the 16-term model accounts for all the leakage terms. It is typically used for measurements made at higher frequencies. Care must be taken when using this model, as movement of the probe positions can cause the leakage terms to change.

In generic calibration, polynomials are proposed to describe the equations and inequalities that make up the calibration. Most two-port calibrations can be described by continuous polynomials and can be defined by their coefficients in a coefficient matrix. Restricting the generic calibration to polynomial equations and inequalities excludes those which use trigonometric functions, such as those involving a phase reference for the calibration of complex components such as distributed or network elements.



CUT: Component under Test

FIGURE 18.20

Model of measurement head. X represents the error model for port 1 and Y represents the error model for port 2.

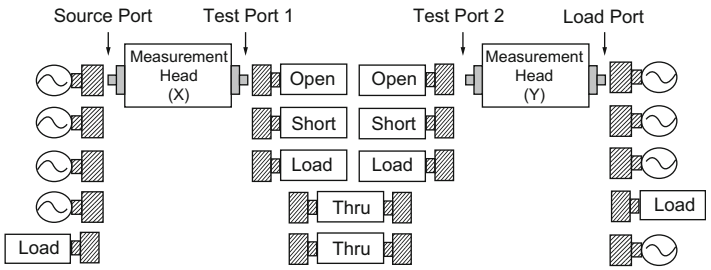


FIGURE 18.21

Procedure for the example as "Unknown Thru" calibration. A known open, known short and a known load are measured at port 1 and port 2. A thru with unknown characteristic is then measured.

This is quite deliberate, because trigonometric functions are difficult to represent without using a symbolic interpreter, and the solution space will typically have many critical points, making it difficult to find a global solution. To overcome this limitation in distributed or network components, the normalizing coefficient (e_{01} in this example) must be calculated in the firmware and the remaining calibration specified in terms of polynomials.

As an example of using an error model in calibration, the "Unknown Thru" is discussed, in which the calibration uses the 8-term error model shown in Figure 18.20. The coefficients of the model (e_{00}, e_{11}, e_{22}, e_{33}, $e_{01}e_{10}$, $e_{23}e_{32}$ and $e_{10}e_{32}$) are calculated by first carrying out a 1-port calibration at each test port using a short, open and load with known reflection coefficients as shown in Figure 18.21. This leads to the following six, nonlinear equations shown in Table 18.2, where F_{C1}, F_{C2} … F_{C6} are the known reflection coefficients of the calibration standards and F_{M1}, F_{M2} … F_{M6} the corresponding measurements.

Solving this group of nonlinear equations leads to solutions for error models consisting of the error terms. The polynomial programming problem can also be solved algebraically by identifying all the critical points and choosing the smallest value of the function at those points. Those readers who have an interest in solving this group of nonlinear equations can look at the Further Reading for methods and tools.

18.4.2 Calibration methods

The goal of calibration is to quantify and to improve the accuracy of an instrument or component used in industrial systems. Technically, calibration is the comparison of a physical measurement of an

Table 18.2 The Error-Model Equations for an Example of "Unknown Thru" Calibration.

$e_{00} + F_{C1}F_{M1}e_{11} - F_{C1}e_{00}e_{11} + F_{C1}(e_{01}e_{10})$	F_{M1}	(1)
$e_{00} + F_{C2}F_{M2}e_{11} - F_{C2}e_{00}e_{11} + F_{C2}(e_{01}e_{10})$	F_{M2}	(2)
$e_{00} + F_{C3}F_{M3}e_{11} - F_{C3}e_{00}e_{11} + F_{C3}(e_{01}e_{10})$	F_{M3}	(3)
$e_{33} + F_{C4}F_{M4}e_{22} - F_{C4}e_{22}e_{33} + F_{C4}(e_{23}e_{32})$	F_{M4}	(4)
$e_{33} + F_{C5}F_{M5}e_{22} - F_{C5}e_{22}e_{33} + F_{C5}(e_{23}e_{32})$	F_{M5}	(5)
$e_{33} + F_{C6}F_{M6}e_{22} - F_{C6}e_{22}e_{33} + F_{C6}(e_{23}e_{32})$	F_{M6}	(6)

instrument with a standard of known accuracy. If performing a dynamic calibration while the system is running, the standards are often numbers stored in the system's memories such as NVM. Application programs for detecting errors may use the comparison between the measurement and the stored standard values. If this calibration does not find error, the application program continues to run. For each component, which requires periodic calibration, the application program sets up a timer to schedule it. When this timer expires, application programs immediately notify the master controller microprocessor to execute the calibration again. So ensuring the accuracy of the calibration basically involves two steps: (1) comparison and (2) periodic checking. The methodologies for maintaining appropriate calibrations depend on the following.

(1) Frequency

Instruments or components should be adjusted periodically while the system is running. Any errors found should be reported to the appropriate master controllers or the system's motherboard for calibration routines to be executed. For example, the scanner of any modern copy machine should be calibrated every 1 5 minutes, depending on whether the scanner is color or black/white, while the machine is running.

The calibration frequency, or time interval, is key to ensuring the performance of an industrial system. If it has some instruments or components requiring periodic calibrations while the system is running, its application programs must be designed to schedule them accordingly.

In addition, the time interval for recalibrating other instruments or components should be determined when it is at rest. Initial consideration for this kind of time interval includes (1) the manufacturers' recommendations, (2) the accuracy sought, and (3) environmental influences. To set it, the instruments' documentation should be checked.

(2) Accuracy

The result of any measurement is only an estimate of the "real" value being measured. In truth, the real value can never be perfectly measured. This is because there are always some physical limits or constraints to how well we can measure a property. For example, a heat sterilization temperature may be determined in the laboratory, and then monitored during the manufacturing stage of a process. Instruments used in either or both locations may indicate temperature to a small fraction of a degree. If not correctly calibrated or are subject to drift, there may be failure of the process because of inaccuracy. Precision often brings a false sense of accuracy.

"Standards" of physical properties such as temperature, pressure, speed, torque, light strength, weight and color scales are the key to ensuring accuracy. Two types are important: primary standards and secondary or reference standards. "Reference" is a term used in two ways in the measurement of physical values. First, it is used to describe the process of comparing the reading of one instrument with another; most commonly the reading from an instrument under calibration with the "known" physical value of a primary standard material or measurement meter. Second, it is used as a term describing a measurement meter itself, to indicate whether it is a master reference or secondary reference. Either way, the term reference refers to a comparison process by which correct calibration is ensured.

Many systems will average the returned data and report this average as the measurement. To determine the statistical uncertainty, you will need to calculate the standard deviation of all of the measurements and include this value as part of the overall measurement accuracy. In metrology, these

accuracies are referred to as type A and type B uncertainties. Type A uncertainties are those due to statistical methods, and type B uncertainties are systematic (gain, offset, etc.).

(3) Traceability

Traceability is the unbroken chain of comparisons between your measurement device and national or international standards. Different legal metrology authorities exist for each country. These bodies follow guidelines defined by the international metrology body and its associated committees to provide quality measurement standards for their country. The National Metrology Institutes (NMI) of each member country of the Convention of the Meter also participate in the Mutual Recognition Agreement (MRA). These international and national metrology organizations serve industry with (1) their calibration services, including the standards and code, (2) their calibration tools, including hardware and software, and (3) the issuing of their calibration certificates. These topics regarding traceability are beyond the scope of this book, and are not mentioned further here.

18.4.3 Calibration techniques

Calibration involves sophisticated theories, complex techniques, and advanced equipment. Different industrial systems and processes need totally different calibration techniques. Furthermore, different calibration techniques are often required in an industrial system and process for different equipment and procedures. Therefore calibration should be based on the basic calibration techniques for elementary physical variables such as temperature, pressure, speed, position, and scale.

In the following we will discuss the calibration of basic physical variables. Due to limited space, this discussion will be very concise.

(1) Temperature calibrations

Temperature is one of the most frequently measured parameters in industrial processes. A wide variety of mechanical and electrical thermometers are used to sense and control process temperature. Regular calibration of these thermometers is critical to ensure consistent quality of the product manufactured, as well as providing regulatory compliance in some industries.

Simply stated, temperature calibration consists of placing the thermometer under test into a known, stable temperature environment. A comparison is then made between the actual temperature and the reading indicated by the thermometer under test and the difference is noted.

Adjustments can then be made either directly to the thermometer or to its readout. Electrical thermometers are adjusted by mathematically recreating the coefficients used by SMART transmitters or other readout devices to translate electrical output to a temperature value. Many mechanical thermometers, such as dial gauges, can be adjusted by turning a dial or other mechanical device. In some cases, such as liquid-in-glass thermometers, direct adjustments are not possible and offsets must be noted.

In industrial applications, the temperature environment is usually provided by a drywell, or dry-block calibrator, or a micro-bath. Both offer portability and a wide range of temperatures. Drywells use high-stability metal blocks with drilled wells to accept the reference and the component under test. Drywells typically cover ranges from $-45°C$ to $1200°C$ and micro-baths cover ranges from $-25°C$ to $200°C$. Micro-baths are similar in size to drywells but use a small tank of stirred fluid instead of a metal block. Micro-baths offer significant advantages when calibrating short or oddly shaped probes.

The "actual" temperature of the bath or dry-well is determined by a reference thermometer, which may be either a thermometer internal to the heat source, or an external reference thermometer operating independently.

(2) Pressure calibrations

There are three classic calibrations for pressure equipment in industrial processes: pressure gauges, switches, and transmitters. The first issue we have to deal with when calibrating pressure equipment is the unit of measure we are dealing with, as there seem to be dozens of them. In Western countries, for example in the USA, pounds per square inch (psi) is usually used but needs to be specified whether this is absolute or gauge pressure (psia or psig). Although psig is usual, inches of water is often used for low-pressure applications. Other units of pressure are: bar, Pascal, inches of mercury, millimeters of water, micron, torr, and dynes per square centimeter.

Many modern pressure calibrations allow selection of the desired units of measure, but we still need to understand the relative pressure we are dealing with to ensure the correct test equipment, pressure module, tubing and fittings rated for the maximum test pressure are used. Figure 18.22 shows the function blocks used for typical pressure calibration test set-ups.

Another challenge with pressure calibrations is to have the correct fittings to connect the pressure source, pressure standard, and the component or unit under test. Even with a few drawers full of fittings in the workshop, and access to more adapters in the mechanical workshop and spare parts stock, we never seem to have the right fittings. To minimize this problem, the required fittings should be added to the calibration notes. The number of connections in the test set-up should be minimized in order to minimize the potential for leakage.

Leakage is another problem we have to deal with when performing pressure calibrations, made worse when a hand pump is used as the pressure/vacuum source. Small leaks in the test set-up can be compensated for by using a constant pressure source such as a N_2 bottle, a spare plant air connection, or a vacuum pump. However, leaks should be eliminated or minimized for all pressure calibrations

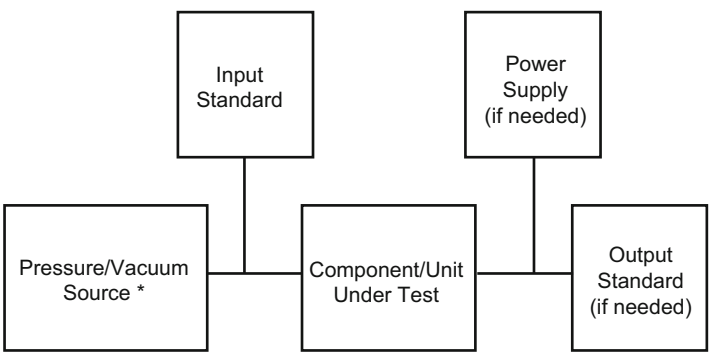


FIGURE 18.22

Typical pressure calibration test set-up.

even when using such a source. Even the smallest leak is frustrating when using a hand pump, so those connections have to be minimized and tight.

Hysteresis is the measure of the difference in response to a device as the input signal increases from a minimum to a maximum value, and, subsequently decreases from maximum to minimum over the same range. It is expressed as percentage of full scale (%FS). Equipment with mechanical movements such as pressure gauges and current-to-pneumatic (I/P) transducers develop hysteresis error, often from friction and wear of the mechanical components. There is no way to adjust for hysteresis error during calibration. It can be evaluated by obtaining calibration data with the input applied in both the increasing and decreasing direction.

When evaluating hysteresis, it is important to approach the target value without overshooting the applied input. If the input value exceeds the increasing target value, decrease the applied input to the previous test point and retry. If the input value exceeds the decreasing target value, increase the applied input above the previous test point and retry.

Special safety issues must be considered when calibrating pressure equipment. First of all, ensure the system pressure has been released prior to attempting to remove any pressure devices. Also, you must know what potential hazards exist with the industrial process material. Some may be toxic or may cause harm to the environment if released. In these cases, verify whether the area is often needed before removing any device for calibration.

(3) Speed calibrations

There are two types of mechanical movements solid and fluid both of which may need calibration of the instruments measuring their speeds. Typical examples of solid speed calibration are the speed-cameras and speedometers of vehicles such as cars, lorries, and trains. In contrast, fluid speeds, for instance water flow rates, are much more difficult to measure precisely, hence more complex calibration techniques are required. In the following we will discuss mention flowmeter calibration, and leave speed-camera and speedometer calibrations to those readers who are interested in them.

All flowmeters with moving parts require periodic testing, since wear over time will affect performance. Calibration can be performed either in the laboratory or at the manufacturers by using a prover, also called a mater meter, or by weighing the flow output. There are several methodologies for flowmeter calibration, any one of which may be acceptable depending on the process system configuration, compatibility, availability of test standards, and accuracy requirements.

It is sometimes difficult or impossible to remove a flowmeter from service for calibration, so field-mounted and inline provers have been developed. Depending on the application and system configuration, other methods can also be applied to check the accuracy of flowmeters. Weighing the flowmeter output collected over a specified time is a commonly used alternative.

The calibration of the signal-processing portion for most flowmeter can be checked by simulating the signal from it. These methods do not check the sensor itself. No one generic method works for all flowmeters. Tests must be performed in accordance with the specific manufacturer's instructions. However, a few of these methods are discussed in Table 18.3.

(4) Position calibrations

Position calibrations are required by many industrial control devices, such as actuators, motors, valves, servos, gears and, in particular, various robots. The positioning accuracy of industrial robots varies by

Table 18.3 Some Calibration Methods for Flowmeters

General Methodology	When to Use
Calibrate only the electronics (or signal processing) using a test instrument to simulate the flowmeter sensor.	If it is impossible to perform a manufacturer check of the flowmeter sensor using a prover or other methods discussed below and the sensor cannot be removed from the system.
Check the calibration of the flowmeter, sensor and signal processing together, using a prover or some other standard.	If the flowmeter sensor can be checked by its manufacturer but test standards are not available for simulating test signal input.
Calibrate the electronics first, and then check the calibration of the flowmeter, including its sensor.	If the required test instruments are available to simulate the flowmeter input signal and the flowmeter sensor can be checked by its manufacturer.
Remove the flowmeter and send to a flow calibration laboratory or its manufacturer.	If the system is not compatible with an inline test, it may be necessary to install a calibrated spare to keep the process downtime to a minimum.

manufacturer, age, and robot type. The magnitude of the error between the actual position and the desired position can be as low as a tenth of a millimeter, or as high as several centimeters.

At the most general level, robot calibration can be classified into two groups. Model-based parametric calibration and model non-parametric calibration. Most work on model-based parametric calibration has concentrated on kinematic calibration. Using this approach, errors between the actual and desired position can often be reduced to less than a millimeter.

A calibrated robot has a higher absolute positioning accuracy than an un-calibrated one, i.e., the real position of the robot end effector corresponds better to the position calculated from the mathematical model of the robot. Absolute positioning accuracy is particularly relevant to robot exchangeability and offline programming of precision applications. As well as calibrating the robot, calibration of its tools and the workpieces it works with (so-called cell calibration) can minimize the occurrence of inaccuracies and improve process security.

Nowadays calibration plays an increasingly important role in robot production as well as in implementation and operation within computer-integrated manufacturing or assembly systems. The production, implementation and operation of robots are all areas where calibration results can lead to significant accuracy improvement and/or cost-saving as explained in the relevant literature.

(5) Scale calibrations

Scale calibration can include calibrating either weight or color scales. It ensures that a weight or a color scale is providing accurate information. For scales used commercially, calibration may be necessary every few weeks or months, to confirm that the scales are still accurate, while home scales may be left uncalibrated for much longer periods.

The best way to perform scale calibration is to use a known weight range or color spectrum to see whether the scale returns the correct measurement. Several companies manufacture standards specifically for use in scale calibration. To calibrate the scale, the weight or color is set on the scale and the reading or checking is noted. Next, the scale can be adjusted until it yields a correct measurement.

Digital scales may have buttons which are used specifically in scale calibration. The vendor manual for a scale usually provides specific information on calibration.

Problems

1. Based on our discussion in this chapter, please give the essential difference between the system operation routines and the kernel/nucleus of the operating system, say, in a PC computer.
2. Explain what firmware is and give one or two examples.
3. Is the BIOS firmware?
4. Investigate the booting process for multicore processor computers (such as those in quad core processor Dell laptops) and answer: (1) does each core processor run its own BIOS or do multicore processors run a common BIOS? (2) Is only one core processor responsible for initializing all hardware components? (3) At the completion of the initializations, how is the operating system loaded?
5. Think about why, if the electrical power connecting to a computer is accidentally cut off, the hard drive of the computer is more easily broken; and when it is restarted, why most computers can work normally after the broken hard drive has been properly repaired.
6. Embedded processors in system on chip architecture includes two types: microprocessors and digital signal processors (DSP). Please investigate the structures of the DSP subsystem and explain how it is different from the microprocessor subsystem.
7. Please identify the intellectual property cores in Figure 18.3 then say which of them are programmable cores and which are non programmable cores.
8. In the exhaustive approach to generate the stimuli in Figure 18.4, the test length is 2^N where N is the number of inputs to the circuit. Please explain why the number of all detectable faults relating to N inputs in an integrated circuit is equal to 2^N.
9. Please give the working method and sequence of two hardware based self test routines for system on chip: the ad hoc test and the scan based test.
10. An example of a pseudorandom pattern generator of stimuli (Figure 18.4) is the linear feedback shift register (or LFSR), which is shown in Figure 18.5(a). A LFSR consists of a serial connection of 1 bit registers. An N bit LFSR cycles through 2^N 1 states before repeating the sequence. Please explain why the serial patterns produced in an N bit LFSR cycles are random.
11. Figure 18.6 depicts the basic concept of embedded processor testing by executing software based self test routines. Please research the working steps in software based embedded processor self test routines.
12. Please give the step by step sequence of two software based self test routines for system on chip: stuck at fault testing and delay fault testing.
13. Why should all the internal buses, including the system buses, in bus based embedded control systems be firmware rather than simply hardware?
14. In Figure 18.7, both the PCI bus 0 and the PCI bus 1 can be categorized as “processor or local bus” or “global or system bus”? Can the PCI bus connecting the super I/O controller be categorized as the I/O bus?
15. PCI device is a vague term for all devices connecting to a bus based embedded system via a PCI slot. Please specify what PCI devices are in your understanding.
16. PCI address and configuration spaces are the control and status registers in the memory or memory spaces shared between the CPU and all the connecting PCI devices. PCI address or configuration spaces offer each PCI device a configuration header, as shown in Figure 18.8 and Table 18.1. Please investigate which agents in a PCI based embedded system are responsible for writing and modifying the configuration header of each PCI device.
17. What is the device driver in an embedded system? What is the device driver in a bus based embedded system? What is the device driver in a PCI based embedded system?
18. Each PCI device has its PCI I/O space and PCI memory space. Most PCIs use a memory mapping technique to link the PCI I/O space and PCI memory space of a PCI device. Is there another technique to link the PCI I/O space and PCI memory space of a PCI device?
19. Using PCI PCI bridges can add more PCI buses to allow the system to support many more PCI devices. Please write a short essay to explain the working principle of PCI PCI bridges.

20. In a PCI based embedded system, is the PCI BIOS the same thing as the CPU BIOS, or are they different?
21. In an industrial control system, the PCI initialization code can be broken into three logical parts: PCI device driver, PCI BIOS, and PCI firmware. Figures 18.10, 18.11, 18.12, and 18.13 describe a process in which a PCI bus system of four bridges is being initialized with the Linux approach. Please write what each of these three logic parts does in each of these four figures.
22. If possible, please investigate the working principles of other types of system bus, such as SCSI and USB, with regard to device installation and configuration in a bus based embedded system.
23. Which approach, top down or bottom up, is mostly used in system diagnosis and in process diagnosis?
24. Please list the roles of fault databases, including fault table and dictionary, in the combinational fault diagnosis method.
25. Can you design a fault database of different format from those given in Figure 18.15 and 18.16 to reduce the large computational effort involved in building a fault dictionary and achieve the maximal fault resolution?
26. Figure 18.17 is a diagnostic tree used for locating faults by edge pin testing. Please find both the starting and final fault nodes to write the scenario of all sequential steps between the starting and final fault nodes in this figure.
27. In Figure 18.19, a diagnostic tree can be created for the given test pattern to control the process of probing. This diagnosis tree consists of internal nodes (circles) to mark the internal lines to be probed, and of terminal nodes (rectangles) to show the possible result of diagnosis. The results of probing are indicated as passed (P) or failed (F). The most time consuming part of guided probe testing is moving the probe. To speed up the fault location process, we need to reduce the number of probed lines. A lot of methods to minimize the number of probes are available. Please investigate as many methods minimizing the number of probes as possible.
28. Try to find a copier or printer such as one made by Hewlett Packard, and let it be idle and observe for half an hour to see whether it is running and how to run calibrations.
29. Try to find a garage to investigate the methods and tools of calibrating a car speedometer; and set up an 8 term error model for it.
30. The generic calibration description consists of an error model and a number of equations and inequalities which describe the relationship between the model coefficients and a number of measurements. Unlike a conventional calibration, the method of solving the system of nonlinear equations and inequalities is not specified. Therefore, the generic calibration is rather more sophisticated than conventional calibration principles. If you have an interest, you can read some of the literature to obtain a deeper understanding of generic calibrations.
31. Solve the group of nonlinear equations given in Table 18.2 for the solutions of error models consisting of the error terms algebraically by identifying all the critical points and choosing the smallest value of the function at those critical points.
32. Please draw a diagram similar to Figure 18.22 for a typical temperature calibration test setup, based on this statement. "Most simply stated, temperature calibration consists of placing a thermometer under test into a known, stable temperature environment. A comparison is made between the actual temperature and the reading indicated by the thermometer under test and the difference is noted."
33. There are three classic calibrations for pressure equipment in industrial processes: pressure gauges, switches, and transmitters. What is the importance of checking the calibration of a pressure gauge while increasing and decreasing pressure is applied? What is the importance of checking the calibration of a pressure switch while increasing and decreasing pressure is applied? What is the importance of checking the calibration of a pressure transmitter while increasing and decreasing pressure is applied?

Further Reading

Angela Krstic, Wei Cheng Lai, Li Chen, Sujit Dey. Self Test for Programmable Core Based Designs. IEEE Designs & Test of Computers. 2 (2002), 18 27.

Dimitris Gizopoulos, Antonis Paschalis, Yervant Zorian. Embedded Processor Based Self Test. Kluwer Academic. 2004.

Erik Larsson. Introduction to Advanced System on Chip test design and Optimization. Springer. 2005.

Wael Badawy, Graham A. Jullien (Eds.). System on Chip for Real time Applications. Kluwer Academic. 2003.
Wikipedia (http://en.wikipedia.org). Booting. http://en.wikipedia.org/wiki/Booting. Accessed: October 2009.
Wikipedia (http://en.wikipedia.org). Firmware. http://en.wikipedia.org/wiki/Firmware. Accessed: October 2009.
Wikipedia (http://en.wikipedia.org). Robot calibration. http://en.wikipedia.org/wiki/Robot calibration. Accessed: October 2009.
GlobalSpec (www.globalspec.com). Software based self testing. http://www.globalspec.com/reference/39755/203279/Chapter 11 Software Based Self Testing. Accessed: October 2009.
GlobalSpec (www.globalspec.com). Products/services for fault diagnosis. http://search.globalspec.com/ProductFinder/FindProducts?query=Products%2FServices%20for%20fault%20diagnosis. Accessed: November 2009.
Sarmad J. Dahir. Functional self test of DSP cores in a SoC. http://www.essays.se/essay/80a5f57898/. Accessed: October 2009.
Dario L. Sancho Pradel, Roger M. Goodall. System on chip (SoC) design for embedded real time control applications. http://www.lboro.ac.uk/departments/el/research/scg/pdfs/sancho.pdf. Accessed: October 2009.
A. Gambier. Real time Control Systems: A Tutorial. IEEE Control Conference, July 20 23 2004; Tokyo. 5th Asian Volume 2; pp. 1024 1031.
Spyros G. Tzafestas, J. K. Pal (Eds.). Real time microcomputer control of industrial processes. Kluwer Academic. 1990.
J. W. S. Liu. Real time Systems. Prentice Hall. 2000.
N. Alexandridis. Computer Systems Architecture: Microprocessor Based Designs. Computer Science Press. 2000.
Fault Diagnosis. http://www.pld.ttu.ee/diagnostika/theory/faultdiagnosis.html. Accessed: November 2009.
David A Rusling. The Linux kernel. http://tldp.org/LDP/tlk/tlk toc.html. Accessed: November 2009.
David A Rusling. PCI. http://tldp.org/LDP/tlk/dd/pci.html. Accessed: November 2009.
Venkat Venkatasubramanian, et al. A review of process fault detection and diagnosis Part I: Quantitative model based methods. Computers and Chemical Engineering, 27 (2003), 293 311.
Venkat Venkatasubramanian, et al. A review of process fault detection and diagnosis Part II: Qualitative models and search strategies. Computers and Chemical Engineering, 27 (2003), 313 326.
Venkat Venkatasubramanian, et al. A review of process fault detection and diagnosis Part III: Process history based methods. Computers and Chemical Engineering, 27 (2003), 327 346.
Rolf Isermann. Fault Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance. Springer. 2005.
Peter Zoeteweij, et al. Automated fault diagnosis in embedded systems. http://www.st.ewi.tudelft.nl/~peterz/papers/ZPAFG BCES07.pdf. Accessed: November 2009.
Ahmad T. Al Taani. An Expert System for Car Failure Diagnosis. World Academy of Science, Engineering and Technology, 12 (2005), 4 7.
Mike Cable. Calibration: A Technician's Guide. ISA. 2005.
Peter S Blockley and James G. Rathmell. Towards generic calibration. http://www.ee.usyd.edu.au/~jimr/pubs/papers/arftg05.pdf. Accessed: November 2009.
Bernard Morris. Temperature calibration in industrial processes. http://www.coleparmer.com/techinfo/techinfo.asp?htmlfile=Ind TempCalibProcess.htm&ID=358#anchor0. Accessed: November 2009.
Fluke. Temperature calibration. http://assets.fluke.com/AppNotes/ProcessTools/1560369 .pdf. Accessed: November 2009.
Omega (www.omega.com). Pressure gauges & switches. http://www.tcomega.com/literature/transactions/volume3/pressure3.html# top. Accessed: November 2009.
Basic colour scale calibration. http://www.avsforum.com/avs vb/showthread.php?t=852536. Accessed: November 2009.
How to do weight scale calibration. http://www.ehow.com/how 4677570 calibrate digital scale.html. Accessed: November 209.

CHAPTER

Industrial control system simulation routines

19

Perhaps one of the greatest opportunities to reduce operating expenses and enhance overall performance without investing additional effort is to first simulate the industrial processes and systems under control or to be controlled. In fact, most industrial control designs rely on simulation of the process and system, which allows the best control strategies and also the best process and system parameters for effective industrial control to be determined. Simulation studies also allow complex what-if scenarios to be viewed, which may be difficult with the real process and system.

However, it must always be borne in mind that any simulation results obtained are only as good as the model of the process or system that is used to obtain them. This does not mean that every effort should be made to get the model as realistic as possible, just that it should be sufficiently representative to produce good enough results.

Creating such a model of an industrial process or system, whatever its type and complexity, requires all features to be described in a form that can be analyzed.

Industrial processes can be classified as discrete or continuous. Industrial systems are embedded or distributed. A key problem in modeling and simulation lies in hybrid processes or systems which require special numerical procedures and integration algorithms. These advanced control strategies can be obtained by experimental approaches to the system and process modeling, so-called "process or system identification". Additionally, the interplay between modeling and control has led to a wide variety of iterative modeling and control strategies, in which control-oriented identification is interleaved with control analysis and design, aiming at the gradual improvement of industrial process controller performance.

Two industrial simulation methods have been developed: computer-direct and numerical-computation. Computer-direct simulation is a fast and flexible tool for generating models, either for current systems where modifications are planned, or for completely new systems. Running the simulation predicts the effect of changing system parameters, provides information on the sensitivity of the system, and helps to identify an optimum solution for specific operating conditions. Numerical-computation simulation is the key to comprehending and controlling the full-scale industrial plant used in the chemical, oil, gas and electrical power industries. Simulation of these industrial processes uses the laws of physics and chemistry to produce mathematical equations to dynamically simulate all the most important unit operations found in process and power plants.

The most important topics to be addressed with respect to simulation routines include modeling and identification; control and simulation; simulators and tools.

Advanced Industrial Control Technology. DOI: 10.1016/B978-1-4377-7807-6.10019-1

19.1 MODELING AND IDENTIFICATION

Industrial control can be split into process and system control, which divides the modeling of industrial control into process modeling and system modeling accordingly.

Process modeling is the mathematical representation of a process by application of material properties and physical laws governing geometry, dynamics, heat and fluid flow, and so on, in order to predict its behavior. For example, finite element analyses are used to represent the application of forces (mechanics, strength of materials) on a defined part (geometry and material properties), and to model a metal forging operation. The result of the analyses is a time-based series of pictures, showing the distribution of stresses and strains, which depict the configuration and state of the part during and after forging. The behavior predicted by process models is compared with the results of actual processes to ensure that the models are correct. As differences between theoretical and actual behavior are resolved, the basic understanding of the process improves and future process decisions are more informed. The analysis can be used to iterate tooling designs and make processing decisions without incurring the high costs of physical prototyping.

System modeling is for a system typically composed of a number of networks that connect its different nodes, and where the networks are interconnected through gateways. One example of such a system is an automotive system that include a high-speed network (based on the controller area network) for connecting engine, transmission, and brake-related nodes; and one network for connecting other "body electronics functions" from instrument panels to alarms. Often, a separate network is available for diagnostics. A node typically includes sensor and actuator interfaces, a microcontroller, and a communication interface to the broadcast (analog or digital) bus. From a control function perspective, the vehicle can be controlled by a hierarchical system, where subfunctions interact with each other and through the vehicle dynamics to provide the desired control performance. The subfunctions are implemented in various nodes of the vehicle, but not always in a top-down fashion, because the development is strongly governed by aspects such as the organizational structure (internal organization, system integrators, and subcontractors).

Models and simulation features should form part of a larger toolset that supports the design of control systems to meet the main identified challenges: complexity, multidisciplinarity, and dependability. Some of the requirements of the models are as follows:

1. The developed system models should encompass both time- and event-triggered algorithms, as typified by discrete-time control and finite-state machines (hybrid systems).
2. The models must represent the basic mechanisms and algorithms that affect the overall system timing behavior.
3. The models should allow co-simulation of functionality, as implemented in a computer system, together with the controlled continuous time processes and the behavior of the computer system.
4. The models should support interdisciplinary design, thus taking into account different supporting methods, modeling views, abstractions and accuracy, as required by control, system, and computer engineers.
5. Preferably, the models should also be useful as a descriptive framework, visualizing different aspects of the system, as well as being useful for other types of analysis such as scheduling analysis.

19.1.1 Industrial process modeling

Any industrial process can be described by a model of that process. In terms of control requirements, the model must contain information that enables the prediction of the consequences of changing process operating conditions. Within this context, a process allows the effects of time and space to be scaled, and extraction of properties and hence simplification, to retain only those details relevant to the problem. The use of models, therefore, reduces the need for real experimentation, and facilitates the achievement of many different purposes at reduced cost, risk, and time.

Depending on the task, different model types will be employed. Process models are categorized as shown in Figure 19.1.

(1) Mathematical models

As specified in Figure 19.1, mathematical models include all of the following:

(a) Mechanistic models

If a process and its characteristics are well defined, a set of differential equations can be used to describe its dynamic behavior. This is known as the development of mechanistic models. The mechanistic model is usually derived from the physics and chemistry governing the process. Depending on the system, the structure of the final model may either be a lumped parameter or a distributed parameter representation.

Lumped parameter models are described by ordinary differential equations, whereas distributed parameter systems representations require the use of partial differential equations. Nevertheless, a distributed model can be approximated by a series of ordinary differential equations and a set of simplifying assumptions. Both lumped and distributed parameter models can be further classified into linear or nonlinear descriptions. Usually nonlinear, the differential equations are often liberalized to enable tractable analysis.

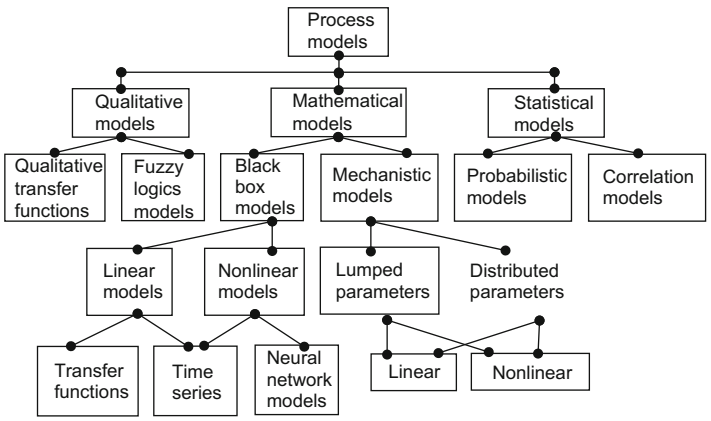


FIGURE 19.1

Classification of the industrial process models.

In many cases, typically due to financial and time constraints, mechanistic model development may not be practically feasible. This is particularly true when knowledge about the process is initially vague, or if the process is so complex that the resulting equations cannot be solved. Under such circumstances, empirical or black-box models may be built, using data collected from the plant.

(b) Black box models

Black box models are simply the functional relationships between system inputs and system outputs. By implication, black box models are lumped together with parameter models. The parameters of these functions do not have any physical significance in terms of equivalence to process parameters such as heat or mass transfer coefficients, reaction kinetics, and so on. This is the disadvantage of black box models compared to mechanistic models. However, if the aim is to merely represent faithfully some trends in process behavior, then the black box modeling approach is very effective.

As shown in Figure 19.1, black box models can be further classified into linear and nonlinear forms. In the linear category, transfer function and time series models predominate. Given the relevant data, a variety of techniques may be used to identify the parameters of linear black box models. Least-squares-based algorithms are, however, the technique most commonly used. Within the nonlinear category, time-series features are found together with neural-network-based models. The parameters of the functions are still linear and thus facilitate identification using least-squares-based techniques. The use of neural networks in model building has increased with the availability of cheap computing power and certain powerful theoretical results.

(2) Qualitative models

There are some cases in which the nature of the process may preclude mathematical description, for example, when the process is operated in distinct operating regions or when physical limits exist. This results in discontinuities that are not amenable to mathematical descriptions. In this case, qualitative models can be formulated. The simplest form of a qualitative model is the rule-based model that makes use of IF THEN ELSE constructs to describe process behavior. These rules are elicited from human experts. Alternatively, genetic algorithms and rule induction techniques can be applied to process data to generate these descriptive rules. More sophisticated approaches make use of qualitative physics theory and its variants. These latter methods aim to rectify the disadvantages of purely rule-based models by invoking some form of algebra so that the preciseness of mathematical modeling approaches can be achieved.

Of these, qualitative transfer functions appear to be the most suitable for process monitoring and control applications, which retain many of the qualities of quantitative transfer functions that describe the relationship between an input and an output variable, particularly the ability to embody temporal aspects of process behavior. The technique was conceived for applications in the process control domain. Cast within an object framework, a model is built up of smaller subsystems and connected together as in a directed graph. Each node in the graph represents a variable while the arcs that connect the nodes describe the influence or relationship between the nodes. Overall system behavior is derived by traversing the graph, from input sources to output sinks.

Fuzzy logic can also be used to build qualitative models. Fuzzy logic theory contains a set of linguistics that facilitates descriptions of complex and ill-defined systems. The magnitudes of changes are quantized as negative medium, positive large, and so on. Fuzzy models are used in everyday life without our being aware of their presence; in for example, washing machines, autofocus cameras, and so on.

(3) Statistical models

Describing processes in statistical terms is another modeling technique. Time-series analysis that has a heavy statistical bias may be considered to fall into this category. Statistical models do not capture system dynamics, but in modern control practice, they play an important role, particularly in assisting in higher-level decision making, process monitoring, data analysis, and, obviously, in statistical process control.

Given its widespread and interchangeable use in the development of deterministic as well as stochastic digital control algorithms, the statistical approach is made necessary by the uncertainties surrounding some process systems. This technique has roots in statistical data analysis, information theory, games theory, and the theory of decision systems.

Probabilistic models are characterized by the probability density functions of the variables. The most common is the normal distribution, which provides information about the likelihood of a variable taking on certain values. Multivariate probability density functions can also be formulated, but interpretation becomes difficult when more than two variables are considered. Correlation models arise by quantifying the degree of similarity between two variables by monitoring their variations. This is again quite a commonly used technique, and is implicit when associations between variables are analyzed using regression techniques.

19.1.2 Industrial system modeling

Modeling has been an essential part of industrial control since the 1970s. However, industrial control requires accurate models that are not easy to build. As a compromise, both data-driven modeling and computational intelligence provide additional modeling alternatives for advanced industrial control applications. These alternatives are illustrated in Figure 19.2.

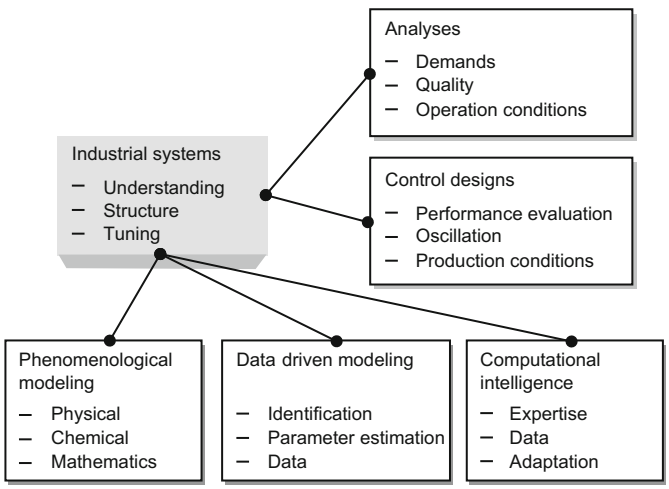


FIGURE 19.2

Alternatives for modeling and simulation of advanced control applications.

(1) Phenomenological modeling approaches

Phenomenological modeling approaches aid in the construction and analysis of models whose ultimate purpose is to provide insights into system performance and improve understanding of how cooperative phenomena can be manipulated for the accomplishment of design strategies that increase the complexity of systems. Following the fundamentals which guide the structuring of the model, the system tasks are decomposed into the relevant phases and physicochemical phenomena involved, identifying the connections and influences among them and evaluating the individual rates. The clarification of rate-limiting phenomena is finally achieved by diverse strategies derived from dealing with the manipulation of driving forces. Empirical models are mostly used for phenomenological modeling to develop a control system. The structure and parameters do not necessarily have any physical significance, and, therefore, these models cannot directly be adapted to different production conditions.

(2) Data-driven modeling approaches

Data-driven modeling approaches are based on general function estimators of a black-box structure, which should capture correctly the dynamics and nonlinearity of the system. The identification procedure, which consists of estimating the parameters of the model, is quite straightforward and easy if appropriate data are available. Essentially, system identification means adjusting parameters within a given model until its output coincides as well as possible with the measured output of the real system. Validation is needed to evaluate the performance of the model.

The generic data-driven modeling procedure consists of the following three steps. The objective of the first step is to define an optimal plant operation mode by performing a model and control analysis. Basic system information, including the operating window and characteristic disturbances, as well as fundamental knowledge of the control structure, such as degrees of freedom of and interactions between basic control loops, will be obtained. The second step identifies a predictive model using data-driven approaches. A suitable model structure will be proposed based on the process dynamics extracted from the operating data, followed by parameter estimation using multivariate statistical techniques. The dynamic partial least squares approach solves the issue of autocorrelation. However, a large number of lagged variables are often required, which might lead to poorly conditioned data matrices. Subspace model identification approaches are suitable to derive a parsimonious model by projecting original process data onto a lower-dimension space that is statistically significant. If a linear model is not sufficient due to strong nonlinearities, neural networks provide a possible solution. The third step is model validation. Independent operating data sets are used to verify the prediction ability of the derived model.

(3) Intelligent modeling methods

Intelligent methods are based on techniques inspired by biological systems and human intelligence, for instance, natural language, neural network rules, semantic network rules, and qualitative models. Most of these techniques have already been used in conventional expert systems.

Computational intelligence can provide additional tools, since humans can handle complex tasks including significant uncertainty on the basis of imprecise and qualitative knowledge. Computational intelligence is the study of the design of intelligent agents. An agent is something that acts in, and affects its environment. They include worms, dogs, thermostats, airplanes, humans, organizations, and

society, and so on. An intelligent agent is a system that acts intelligently: what it does is appropriate to its circumstances and its goal; it is flexible to changing environments and changing goals; it learns from experience; and it makes appropriate choices given perceptual limitations and finite computation. The central goal of computational intelligence is to understand the principles that make intelligent behavior possible, in natural or artificial systems. The main hypothesis is that reasoning is computation. The central engineering goal is to specify methods for the design of useful, intelligent artifacts.

Modeling is also used on other levels of advanced control: high-level control is in many cases based on modeling operator actions, and helps to develop intelligent analyzers and software sensors. The products of modeling are models that are used in adaptive and direct model-based control. Smart adaptive control systems integrate all these features, as shown in Figure 19.3.

Adaptive controllers generally contain two extra components compared to standard controllers. The first is a process monitor, which detects changes in process characteristics either by performance measurement or by parameter estimatation. The second is the adaptation mechanism, which updates the controller parameters. In normal operation, efficient reuse of controllers developed for different operating conditions is good operating practice, as the adaptation always takes time.

An adaptation controller is a controller with adjustable parameters, that can perform online adaptation. However, controllers should also be able to adapt to changing operating conditions in processes where the changes are too fast or too complicated for online adaptation. Therefore, the area of adaptation must be expanded; the adaptation mechanism can be either online or predefined.

(1) Online adaptation

This includes self-tuning, auto-tuning, and self-organization. For online adaptation, changes in process characteristics can be detected through online identification of the system model, or by assessment of the control response, that is, performance analyses. The choice of measure depends on the type of response the control system designer wishes to achieve. Alternative measures include overshoot, rise time, setting time, delay ratio, frequency of oscillations, gain and phase margins, and various error signals.

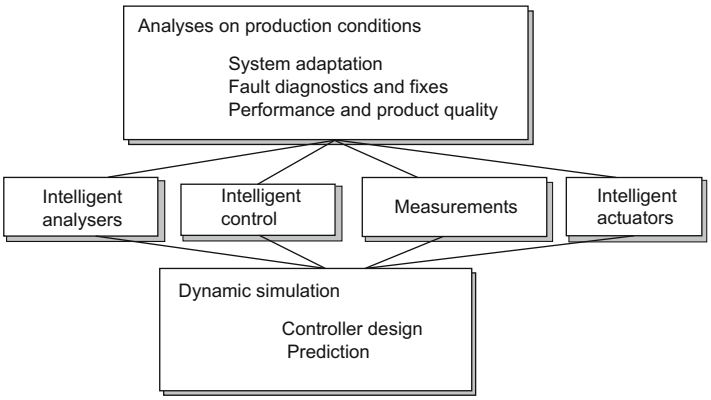


FIGURE 19.3

Features of a smart adaptive control system.

The identification block typically contains some kind of recursive estimation algorithm that aims at the current instant. Figure 19.4 shows online adaptation with model identification. The model can be a transfer function, a discrete-time linear model, a fuzzy model, or a linguistic equation model. Adaptation mechanisms rely on parameter estimations of the process model: gain, dead time, time constants etc.

Classic adaptation schemes do not cope easily with large and fast changes unless the adaptation rate is very high. This is not always possible: some a priori knowledge about the dynamic behavior of a plant or factory may be necessary. One alternative for these cases is a switching control scheme that selects a controller from a finite set of predefined fixed controllers. Multiple-model adaptive control is hence classified as mode-based control. Intelligent methods provide additional techniques for online adaptation. Fuzzy, self-organizing controllers give an example of intelligent modeling methodologies. Another example of this approach is a meta-rule approach, in which parameters of a low-level controller are changed by a meta-rule supervisory system whose decisions are based on the performance of the low-level controller. Meta-rule modules consist typically of a fuzzy rule base that describes the actions needed to improve the low-level fuzzy logic controller.

(2) Predefined adaptation

This is becoming more popular as modeling and simulation techniques improve. Gain scheduling, including fuzzy-gain scheduling and linguistic-equation-based gain scheduling, provides a gradual adaptation technique for a fixed control structure. Predefined adaptation uses very detailed models; for example, distributed parameter models can be used to tune adaptation models of a solar-powered plant. The resulting adaptation models or mechanisms should be able to handle special situations in real time without using detailed simulation models.

Predefined actions adapt rapidly enough to remove the need for online identification, or for classic mechanisms based on performance analysis. In these cases, the controller could be classified as a linguistic equation based on gain scheduling. The adaptation model is generated from the local tuning results, but the directions of interactions are usually consistent with process knowledge.

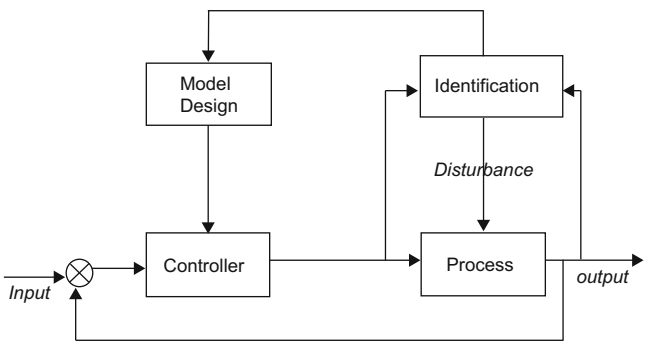


FIGURE 19.4

Online adaptation with model identification.

19.1.3 Identification for model-based control

Process control systems typically include one or more controllers which are communicatively coupled to each other, to at least one host or operator workstation and to one or more field devices via analog, digital or combined buses. Some systems use function blocks or groups of function blocks, referred to as modules to perform control operations.

Industrial process controllers are typically programmed to execute different algorithms, routines or control loops for a process, such as flow, temperature, pressure, or level control loops. Generally speaking, each loop includes one or more input blocks, such as an analog-input, a single-output, or a fuzzy logic control function block, and an output block.

An identification experiment consists of perturbing process or system inputs and observing the responses from the outputs. A model describing this dynamic input-output relationship can then be identified directly by modifying variables and parameters. In a process or system control context, the end-use of such a model would typically be for designing industrial controllers. Ideally, the design of a control-oriented identification procedure could then be formulated by choosing a performance objective, then designing the identification in such a way that the performance achieved by the model-based controller on the true system is as high as possible. Figure 19.4 illustrates this procedure.

In the rest of this subsection, model-based identification for control is introduced very briefly; the mathematical, statistical and physical theories involved in this identification is beyond the scope of this book.

(1) Experiment design

Assuming you need to perform an identification, the first step is to design one or more experiments to capture the process's dynamic behavior. To provide the best results, these experiments must satisfy the following criteria: all modes of process behavior must be represented in the measured data; transient and steady-state behavior must be represented from input to output; the experiments must not damage the plant or cause it to malfunction.

In many cases, such experiments can only be performed on a plant that is in operation. A controller might even be driving the plant (presumably one that is inadequate in some way). The experiment for this type of process would involve adding perturbation signals at the input and measuring the plant's response to the controller's command plus the perturbation.

It is important to remember that such experiments are generally used to develop linear plant models. Input signals must be chosen so the plant behavior remains (at least approximately) linear, meaning that the amplitude of the input signal (and possibly its rate of change) must be limited to a range in which the plant response is approximately linear. However, the amplitude of the driving signal must also be large enough so that corrupting effects such as noise and quantization in the measured output signal are minimized relative to the expected response.

(2) Data collection

The sampling rate for recording input and output data must be high enough to retain all frequencies of interest in the results. According to the Nyquist sampling theorem, the sampling rate (samples per second) must be at least twice the highest frequency of interest to enable reconstruction of the original signal. In practice, the sampling rate must be a little higher than this limit to ensure the resulting data will be useful. For example, if the highest frequency of interest was 20 Hz (hertz), a sampling rate of 50 Hz might be sufficient.

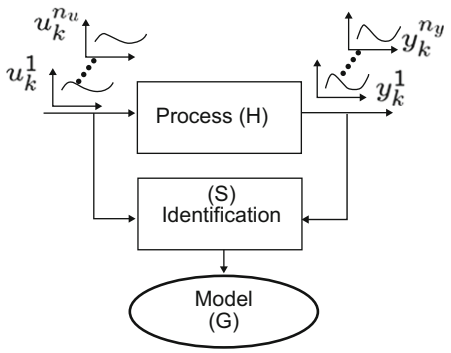
It is also important to keep corrupting effects such as noise and quantization to a minimum relative to the measured signals. The amplitudes of input signals must be sufficient to give large outputs relative to noise and quantization effects. This requirement must be balanced against the need to maintain the system in an approximately linear mode of operation, which typically requires that the input signal amplitudes be small.

(3) Identification calculations

Figure 19.5 gives the basic concept for model-based identification calculations. First, you must have a precise understanding of the process outputs y^j_k. Identification usually starts with introducing a sequence of inputs u^j_k. By testing the time series of y_k with u_k, one model type can be estimated. Repeating the experiment for nearly all model types, the identification calculation is finally to minimize the variance cost, J, of the control performance. Different identification methods result from different formations of this cost.

(4) Iterative identification

The situation described in the above shows that control-relevant models are obtained when identification takes place when a controller is already part of the process. As this controller is unknown before the model is identified, an iterative scheme is required to arrive at the desired situation: (1) perform an identification experiment with the process being controlled by an initial stabilizing controller; (2) identify a model with a control-relevant criterion; (3) design a model-based controller; (4) implement the controller in the process and return to the first step while using the new controller.



$$\min_{(u_k, u_{k+1}, \ldots, u_{k+M-1})} J((u_k, u_{k+1}, \ldots, u_{k+M-1}))$$

$$J = \sum_{j=1}^{P} \left[\|(y_{k+j} - y^r_{k+j})\|_{Q_j} + \|s_j\|_T \right] + \sum_{j=0}^{M-1} \left[\|(u_{k+j} - u_s)\|_{R_j} + \|\Delta u_{k+j}\|_{S_j} \right]$$

FIGURE 19.5

Concept of process identification calculations.

A motivation for applying an iterative scheme is the fact that when designing control systems, the performance limitations are generally not known beforehand. Therefore, a sketched iterative scheme can also be considered to allow improvement of the performance specifications of the controlled system as one learns about the system through dedicated experiments. In this way, improved knowledge of the process dynamics allows the design of a controller with higher performance, thus enhancing overall control performance.

19.2 SIMULATION AND CONTROL

Most control system designs rely on modeling the system to be controlled, allowing simulation studies to determine control strategies and optimum system parameters. These studies also allow complex what-if scenarios to be viewed, which may be difficult on the real system. However, it must always be borne in mind that any simulation results obtained are only as good as the model of the process. This does not mean that every effort should be made to get the model as realistic as possible, just that it should be sufficiently representative.

A simulation is the execution of a process or system model in a program that gives information about what is being investigated. The model consists of events, activated at certain points in time, that affect the overall state of the process or system. Computer simulation is a fast and flexible tool for generating models, either for current systems where modifications are planned or for completely new systems. Running the simulation predicts the effect of changing system parameters, provides information on the sensitivity of the process or system, and helps to identify an optimum solution for specific operating conditions.

The most important topics in modeling and simulation are:

1. Discrete event simulation is applicable to systems whose state changes abruptly in response to some event in the environment, examples being service facilities such as queues at printers and factory production lines;
2. Modeling of real-time systems is used to define requirements and high-level software design before the implementation stage is attempted;
3. Continuous time simulation is used to compute the evolution in time of physical variables such as speed, temperature, voltage, level, position and so on for systems such as robots, chemical reactors, electric motors, and aircraft;
4. Rapid prototyping of control systems involves moving from the design of a controller to its implementation as a prototype.

In this section, the manufacturing process is selected as a representative process control and the computer control system as a representative system control the purposes of this discussion.

19.2.1 Industrial process simulation

In manufacturing processes, modeling and simulation can ensure the best balance of all constraints in designing, developing, producing, and supporting products. Cost, time compression, customer demands, and lifecycle responsibility will all be part of an equation balanced by captured knowledge, online analyses, and human decision-making. Modeling and simulation tools (to be discussed in the

next section) can support best practice from concept creation through product retirement and disposal, and operation of the tools will be transparent to the user. Modeling and simulation activities in the manufacturing process can be divided into five functional elements, as given below.

(1) Material processing

Material processing involves all activities associated with the conversion of raw materials and stock to either a finished form or readiness for assembly.

Enterprise process modeling and simulation environments provide the integrated functionality to ensure the best material or product is produced at the lowest cost. The models include new, reused, and recycled materials to eliminate redundancy. An open, shared industrial knowledge base and model library are required to provide ready access to material properties data using standard forms of information representation (scaleable plug-and-play models), the means for validating models before use, a fundamental science-based understanding of material properties to processes, standard, validated time and cost models and supporting estimating tools for the full range of processes.

Material processing includes four general categories of process, which have a related type of modeling and simulation:

(a) Preparation and creation processes such as synthesis, crystal-growing, mixing, alloying, distilling, casting, pressing, blending, reacting, and molding;
(b) Treatment processes such as coating, plating, painting, and thermal and chemical conditioning;
(c) Forming processes for metals, plastics, composites, and other materials, including bending, extruding, folding, rolling, shearing, stamping;
(d) Removal and addition processes such as milling, drilling, routing, turning, cutting, sanding, trimming, etching, sputtering, vapor deposition, solid freeform fabrication, and ion implantation.

(2) Assemblies, disassembly, and reassembly

This functional element includes processes associated with joining, fastening, soldering, and integration of higher-level packages as required to complete a deliverable product (e.g., electronic packages); it also includes assembly sequencing, error correction and exception handling, disassembly and reassembly, which are maintenance and support issues.

Assembly modeling is well-developed for rigid bodies and tolerance stack-up in limited applications. Rapidly maturing computer-aided design and manufacturing technologies, coupled with advanced modeling and simulation techniques, offer the potential to optimize assembly processes for speed, efficiency, and ease of human interaction. Assembly models will integrate seamlessly with master product and factory operations models to provide all relevant data to drive and control each step of the design and manufacturing process, including tolerance stack-ups, assembly sequences, ergonomics issues, quality, and production rate to support part-to-part assembly, disassembly, and reassembly.

The product assembly model will be a dynamic, living model, adapting in response to changes in requirements and promulgating those changes to all affected elements of the assembly operation including process control, equipment configuration, and product measurement requirements.

(3) Qualities, test, and evaluation

This element includes designs for quality, in-process quality, all inspection and certification processes, such as dimensional, environmental, and chemical and physical property evaluation based on requirements and standards, and diagnostics as well as troubleshooting.

Modeling, simulation, and statistical methods are used to establish control models to which processes should conform. Process characterization, gives models which define the impact of different parameters on product quality. These models are used as a baseline for establishing and maintaining in-control processes. In general, the physics behind these techniques (e.g., radiological testing, ultrasonic evaluation, tomography, and tensile testing) should be well understood. However, many of the interactions are treated probabilistically and, even though models of the fundamental interactions exist, in most cases empirical methods are used instead. Although the method needs to be improved, at present the best way to find out whether a part has a flaw is to test samples. "Models" are used in setting up these experiments, but many times the models reside only in the brains of the experts who support the evaluations.

(4) Packaging

This element includes all final packaging processes, such as wrapping, stamping and marking, palletizing, and packing.

Modeling and simulation are critical for designing packaging that protects products. Logistic models and part tracking systems help us to ensure correct packaging and labeling. Applications range from the proper wrapping for chemical, food, and paper products to shipping containers that protect military hardware and munitions from accidental detonation.

Future process and product modeling and simulation systems will enable packaging designs and processes to be fully integrated into all aspects of the design-to-manufacturing process, and will provide needed functionality at minimum cost, and environmental impact, with no nonvalue-added operations. They will enable designers to optimize packaging designs and supporting processes, to give enhanced product value and performance, as well as for protection, preservation, and handling attributes.

(5) Remanufacture

This includes all design, manufacture, and support processes that support return and reprocessing of products on completion of their original intended use.

Manufacturers can reuse, recycle, and remanufacture products and materials to minimize material and energy consumption, and to maximize the total performance of manufacturing operations. Advanced modeling and simulation capabilities will enable manufacturers to explore and to analyze remanufacturing options to optimize the total product realization process and product and process life cycles for efficiency, cost-effectiveness, profitability, and environmental sensitivity.

Products will be designed from inception for remanufacture and reuse, either at the whole product, or the component or constituent material level. In some cases, ownership of a product may remain with the vendor (not unlike a lease), and the products may be repeatedly upgraded, maintained, and refurbished to extend their lives and add new capabilities.

19.2.2 Industrial system simulation

Modeling and simulation activities in computer control systems could be improved if the following technical issues are taken into account.

(1) Modeling purpose and simulation accuracy

One well-known challenge in modeling is to be able to identify the accuracy that is for a given purpose. Consider, for example, the implementation of a data-flow over two processors and a serial network. A huge span of modeling detail is possible, ranging from a simple delay over discrete-event resource management models (e.g., processor and communication scheduling) to low-level behavioral models. The mapping of these details between models and real computer networks will change the timing behavior of the functions due to effects such as delays and jitter. Some reflections related to this are as follows:

(a) The introduction of application-level effects, such as delays, jitter, and data loss, into a control design could be an appropriate abstraction for control engineering purposes. The mapping to the actual computer system may be nontrivial. For example, delays and jitter can be caused by various combinations of execution, communication, interference, and blocking. More accurate computer system models will be required to compare alternative designs (architectures), and to provide estimations of the system behavior. In addition, modeling is a sort of prototyping, and as such important in the design process.

(b) It is interesting that the underlying model of the computer control system could contain more or less detail, given the right abstraction. For example, if a fairly detailed computer area network (CAN) model has been developed, it could still be used in the context of control system simulation given that it is sufficiently efficient to simulate and that its complexity can be masked off.

(c) It is obvious that models of a computer control system need to reflect the real system. In the early stages, architecture only exists on the drawing board. As the design proceeds, more and more details will be available; consequently, the models used for analysis must be updated accordingly.

(d) To achieve accuracy, close cooperation between software and hardware developers is necessary, and is required at every stage during the system development.

(2) Global synchronization and node tasking

Both synchronous and asynchronous systems exist; industrially asynchronous systems predominate but this may change with the introduction of newer safety-critical applications such as steer-by-wire in cars, because of the advantages inherent in distributed systems based on a global clock.

Under the microscope, communication circuits are typically hard-synchronized, as is necessary to be able to receive bits and arbitrate properly. In a system with low-level synchronization, and/or synchronized clocks, the synchronization could fail in different ways. For asynchronous systems, it could be of interest to incorporate clock drift. Given different clocks with different speeds, this will affect all durations within each node. A conventional way of expressing duration is simply by a time value; in this case the values could possibly be scaled during the simulation set-up. All the above-mentioned behaviors could be interesting to model and simulate.

The most essential characteristic of a distributed computer system is undoubtedly its communication. In early design stages, the distributed and communication aspects are often targeted first; but then node scheduling also becomes interesting and is, therefore, of high relevance. It is very common that many activities coexist on nodes. Typically they have different timing requirements and may also be safety-critical. The scheduling on the nodes affects the distributed system by causing local delays that can influence overall behavior.

When developing a distributed control system, the functions and elements thereof need to be allocated to the nodes. This principally means that an implementation-independent functional design needs to be enhanced with new "system" functions, which for example (1) perform communication between parts of the control system, now residing on different nodes; (2) perform scheduling of the computer system processors and networks; (3) perform additional error-detection and handling to cater for new failure modes (e.g., broken network, temporary node failure, etc.).

A node is composed of application activities, system software including a real-time kernel, low-level I/O drivers, and hardware functions including the communication interface.

(a) The node task model

This needs to include the following: (i) a definition of tasks, their triggers, and execution times for execution units; (ii) a definition of the interactions between tasks in terms of scheduling, inter-task communication, and resource sharing; (iii) a definition of the real-time kernel and other system software with respect to execution time, blocking, and so on. Some issues in the further development include the types of inter-task communication and synchronization that should be supported (e.g., signals, mailboxes, semaphores…) and whether, and to what extent, there is a need to consider hierarchical and hybrid scheduling (e.g., including both the processor's interrupt and real-time kernel scheduling levels).

(b) The functional model

The functional model used in conventional control design should be reusable within the combined-function computer models. This implies that it should be possible to adapt or refine the functional models to incorporate a node-level tasking model.

(c) Communication models

The types of communication protocols are determined by the area under consideration. Nevertheless, a number are currently being developed with a view to future embedded control systems. The CAN network is currently a default standard, but there is also an interest in including the following: (i) time-triggered computer area networks, which refers to CAN systems designed to incorporate clock synchronization suitable for distributed control applications; this rests on the potential of the recent ISO revision of CAN to more easily implement clock synchronization; (ii) properties reflecting state-of-art fault-tolerant protocols such as the time-triggered protocol. The fault-tolerance mechanisms of these protocols, such as membership management and atomic broadcasts, then need to be appropriately modeled.

It is often the case that parts of the protocols are realized in software, for example, dealing with message fragmentation, certain error detection, and potential retransmissions. In this case both the execution of the protocol and the scheduling need to be modeled. The semantics of the communication and in particular of buffers is another important aspect; compare, for example, overwriting and unconsuming semantics versus different types of buffering. Whether this is blocking or not from the point of view of the sender and receiver is also related to its semantics.

Another issue is which "low-level" features of communication controllers need to be taken into account. Compare, for example, the associative filtering capability of CAN controllers, and their internal sorting of message buffers scheduled for transmission (relates back to hierarchical scheduling).

(d) Fault models

The use of fault models is essential for the design of dependable systems. The models of interest are very application-specific, but generic fault models and their implementation are of great interest. There are a number of studies available on fault models dealing with transient and permanent hardware faults, and to some extent also categorizing design faults. As always, there is also the issue of insertion in the form of a fault, an error, or a failure. Consequently, this is a prioritized topic for further work.

(3) System development and tool implementation

While developing distributed control systems, it would be advantageous to have a simulation toolbox or library in which the user can build the system based on prebuilt modules to define things such as the network protocols and the scheduling algorithms. With such a tool, the user could focus on the application details instead. This is possible, since components are standardized across applications and are well defined. In the same way as a programmer works on a certain level of abstraction, at which the hardware and operating system details are hidden by the compiler, the simulation tool should give the user a high level of abstraction to develop the application. Such a tool would enforce a boundary between the application and the rest of the system which would speed up the development process, and gives the developers extra flexibility.

It is clear that the implementation of hybrid systems (such as those combining discrete-event with continuous-time) that we are aiming to model requires a thorough knowledge of the simulation tool. Co-simulation of hybrid systems requires the simulation engine to handle both time-driven and event-triggered parts. The former includes sampled subsystems as well as continuous-time subsystems, handled by a numerical integration algorithm that can be based on a fixed or varying step size. The latter may involve state machines and other forms of event-triggered logic. Some aspects that need consideration for tool implementation are as follows:

(a) If events are used in the computer system model these must be detected by the simulation engine. How is the event-detection mechanism implemented in the simulation tool?
(b) At which points are the actions of a state flow system carried out?
(c) How can actions be defined to be atomic (carried out during one simulation step)?
(d) How can preemption of simulation be implemented, including temporary blocking to model the effects of computer system scheduling?

19.2.3 Industrial control simulation

A control simulation is defined as the reproduction of a situation through the use of process models. For complex control projects, simulation of the process is often a necessary measure for validating the process models to develop the most effective control scheme. Many different types are used in product development today, which can be categorized as comparison or inverse models, as illustrated in Figure 19.6.

Modeling and simulation can be aimed at creating controllers for the processes, which is either a software toolkit or an electronic device. The controllers developed are mostly used for performing so-called model-based control, widely applied in industrial applications. The following paragraphs briefly discuss the various algorithms arising from controller designs in model-based control.

(1) Feed-forward control

Feed-forward control can be based on process models. A feed-forward controller has been combined with different feedback controllers; even the ubiquitous three-term proportional-integral-derivative (PID) controllers can be used for this purpose. A proportional-integral controller is optimal for a first-order linear process (expressed with a first-order linear differential equation) without time delays. Similarly, a PID controller is optimal for a second-order linear process without time delays. The modern approach is to determine the settings of the PID controller based on a model of the process, with the settings chosen so that the controlled responses adhere to user specifications. A typical criterion is that the controlled response should have a quarter decay ratio, or it should follow a defined trajectory, or that the closed loop has certain stability properties.

A more elegant technique is to implement the controller within an adaptive framework. Here the parameters of a linear model are updated regularly to reflect current process characteristics. These parameters are in turn used to calculate the settings of the controller, as shown schematically in Figure 19.7. Theoretically, all model-based controllers can be operated in an adaptive mode, but there are instances when the adaptive mechanism may not be fast enough to capture changes in process characteristics due to system nonlinearities. Under such circumstances, the use of a nonlinear model may be more appropriate. Nonlinear time-series, and neural networks, have been used in this context. A nonlinear PID controller may also be automatically tuned, using an appropriate strategy, by posing the problem as an optimization problem. This may be necessary when the nonlinear dynamics of the plant are time-varying. Again, the strategy is to make use of controller settings most appropriate to the current characteristics of the controlled process.

(2) Model predictive control

Model predictive control (MPC) is an effective means of dealing with large multivariable constrained control problems. The main idea is to choose the control action by repeatedly solving online an optimal control problem, aiming to minimize a performance criterion over a future horizon, possibly subject

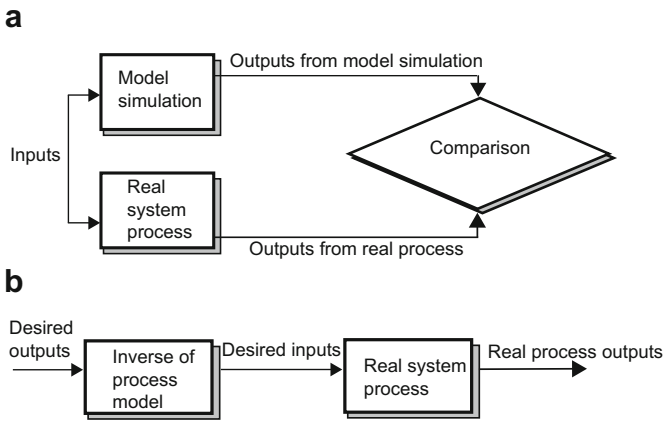


FIGURE 19.6

Classification of process simulations: (a) comparison and (b) inverse model.

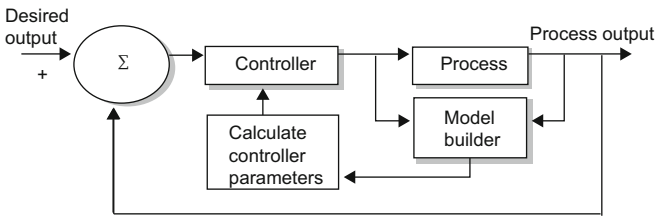


FIGURE 19.7

Schematic of adaptive controllers.

to constraints on the manipulated inputs and outputs. Future behavior is computed according to a model of the plant. Issues arise in guaranteeing closed-loop stability, handling model uncertainty, and reducing online computations.

PID type controllers do not perform well when applied to systems with significant time delays. Model predictive control overcomes the debilitating problems of delayed feedback by using predicted future states of the output for control. Figure 19.8 gives the basic principle of model-based predictive control. Currently, some commercial controllers have Smith predictors as programmable blocks, but there are many other strategies with dead-time compensation properties. If there is no time delay, these algorithms usually collapse to the PID form. Predictive controllers can also be embedded within an adaptive framework, and a typical adaptive predictive control structure is shown in Figure 19.9.

(3) Physical-model-based control

Control has always been concerned with generic techniques that can be applied across a range of physical domains. The design of adaptive or nonadaptive controllers for linear systems requires a representation of the system to be controlled. For example, observer and state-feedback designs require a state-space representation, and polynomial designs require a transfer-function representation. These representations are generic in the sense that they can represent linear systems drawn from a range of physical domains, including mechanical, electrical, hydraulic, and thermodynamic. However, at the same time, these representations suffer from being abstractions of physical systems: the very process of abstraction means that system-specific physical details are lost. Both parameters and states of such representations may not be easily related back to the original system parameters. This loss is, perhaps, acceptable at two extremes of knowledge about the system where the system parameters are completely known, or where they are entirely unknown. In the first case, the system can be translated into the representations mentioned above, and the physical system knowledge is translated into, for example, transfer function parameters. In the second case, the system can be deemed to have one of the representations mentioned above and there is no physical system knowledge to be translated. Thus, much of the current body of control achieves a generic coverage of application areas by having a generic representation of the systems to be controlled, which are, however, not well suited to partially known systems.

This suggests an alternative approach that, whilst achieving a generic coverage of application areas, allows the use of particular representations for particular (possibly partially known, possibly nonlinear) systems. Instead of having a generic representation, a generic method, called

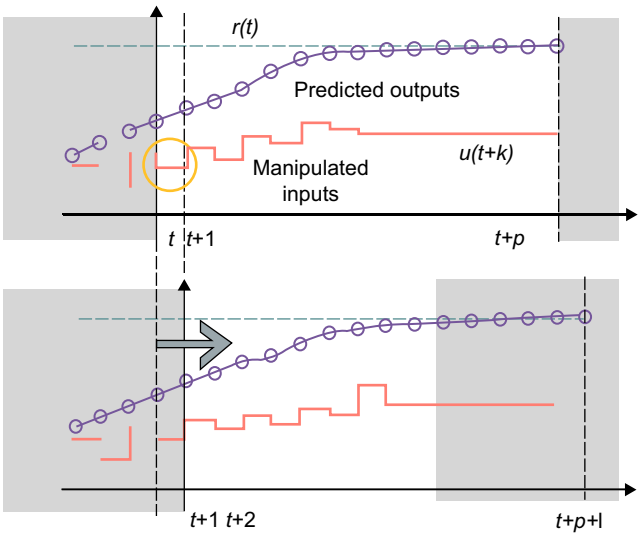


FIGURE 19.8

The basic working principle of model predictive control.

meta-modeling has been proposed; it provides a clear conceptual division between structure and parameters, as a basis for this.

(4) Internal models and robust controls

Internal model control systems are characterized by a control device consisting of the controller and of a simulation of the process, the internal model. The internal model loop computes the difference between the outputs of the process and of the internal model, as shown in Figure 19.10. This difference represents the effect of disturbances and of a mismatch of the model. Internal model control devices

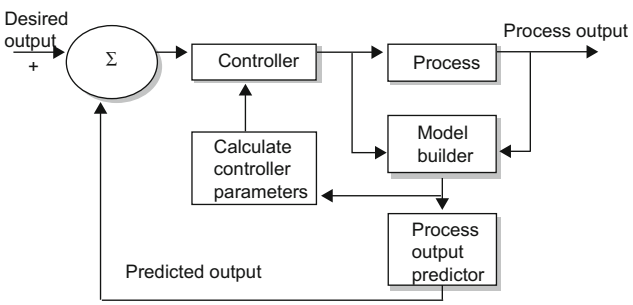


FIGURE 19.9

Schematic of adaptive predictive controllers.

have been shown to be robust against disturbances and model mismatch in the case of a linear model of the process. Internal model control characteristics are the consequence of the following properties.

(a) If the process and the controller are (input-output) stable, and if the internal model is perfect, then the control system is stable.
(b) If the process and the controller are stable, if the internal model is perfect, if the controller is the inverse of the internal model, and if there is no disturbance, then perfect control is achieved.
(c) If the controller steady-state gain is equal to the inverse of the internal model steady-state gain, and if the control system is stable with this controller, then offset-free control is obtained for constant set points and output disturbances.

As a consequence of (c) above, if the controller is made of the inverse of the internal model cascaded with a low-pass filter, and if the control system is stable, then offset-free control is obtained for constant inputs, which are set points and output disturbances. Moreover, the filter introduces robustness against a possible mismatch of the internal model, and, though the gain of the control device without the filter is not infinite as in the continuous-time case, its concern is to smooth out rapidly changing inputs. Robust control involves, first, quantifying the uncertainties or errors in a nominal process model, due to nonlinear or time-varying process behavior, for example. If this can be accomplished, we essentially have a description of the process under all possible operating conditions. The next stage involves the design of a controller that will maintain stability as well as achieve specified performance over this range of operating conditions. A controller with this property is said to be robust.

A sensitive controller is required to achieve performance objectives. Unfortunately, such a controller will also be sensitive to process uncertainties and hence suffer from stability problems. On the other hand, a controller that is insensitive to process uncertainties will have poorer performance characteristics in that controlled responses will be sluggish. The robust control problem is therefore formulated as a compromise between achieving performance and ensuring stability under assumed process uncertainties. Uncertainty descriptions are at best very conservative, whereon performance objectives will have to be sacrificed. Moreover, the resulting optimization problem is frequently not well posed. Thus, although robustness is a desirable property, and the theoretical developments and analysis tools are quite mature, application is hindered by the use of daunting mathematics and the lack of a suitable solution procedure.

Nevertheless, underpinning the design of robust controllers is the so-called internal model principle. It states that unless the control strategy contains, either explicitly or implicitly,

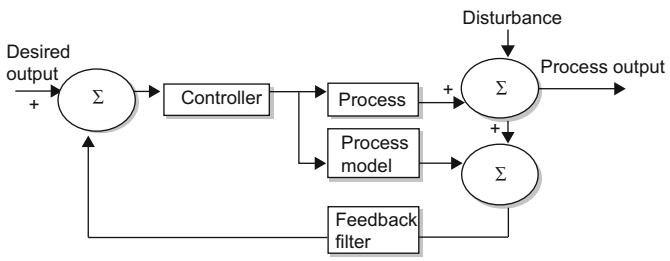


FIGURE 19.10

Strategies of internal model control.

a description of the controlled process, then either the performance or stability criterion, or both, will not be achieved. The corresponding internal model control design procedure encapsulates this philosophy and provides for both perfect control and a mechanism to impart robust properties (see Figure 19.10).

19.3 SOFTWARE AND SIMULATOR

The simulation of both industrial process and system models including controllers must be experimented with by either simulation software or simulators. Simulation software consists of computer programs that can imitate real events and phenomena as realistically as possible based on a group of mathematical equations and formulas, or a set of logical relations and state transfers. Hundreds of such packages have been developed, nevertheless, which can be suitable for such classes as discrete events, continuous processes, single nodes and network systems, and electronic circuits.

An industrial simulator is a special system which integrates computation, visualization, and programming into an easy-to-use environment which provides tools and facilities for simulation hardware and software as well as mechanical instruments. There are a considerable number of industrial simulation tools for various applications, which are therefore impossible to cover in this section. Four of them are briefly introduced in subsection 19.3.2.

19.3.1 Industrial simulation software

In general, industrial simulation software is developed for discrete-event simulation, continuous process or system simulation, network simulation, or electronic circuit simulation. Many industrial simulators, such as CNC and PID controller simulators, are actually effective combinations of different types.

(1) Discrete-event simulation software

A discrete-event simulation is one in which the state of a model changes at only a discrete, but possibly random, set of time points. This often leads to logical complexity because it raises questions about the order in which two or more units are to be manipulated at one time point. For example, two or more traffic units often have to be manipulated at the same time. Such a "simultaneous" movement is achieved by manipulating units of traffic serially at that time point. The challenges faced by a software designer must take the logical requirements of discrete-event simulation into account in a generalized way. Choices and trade-offs exist. As a result, although the software packages are similar in broad terms, they can and typically do differ in subtle but important particulars.

In addition to the representation of state variables and the logical relations of what happens when events occur, discrete-event simulations include the following elements:

(1) Events

An event is a happening that changes the state of a process or a system model. In a model of an order-filling system, for example, the arrival of an order, which is an event, might be simulated by bringing an entity into the model.

(2) Entities

The term entity is used here to designate a unit of transaction. Entities instigate and respond to events. There are two possible types, here referred to as external entities and internal entities. External entities are those whose creation and movement is explicitly arranged for by the modeler. Entities migrate from state to state while they work their way through a model. Correspondingly, simulation software uses the following lists to organize and track entities in the five entity states. (i) The active entity forms an unnamed "list" consisting only of the active entity. (ii) Entities in the ready state are kept in a single list called the current events list. (iii) Entities in the time-delayed state belong to a single list called the future events list. (iv) Delay lists keep tentities in the condition-delayed state. (vi) User-managed lists are lists of entities in the dormant state.

(3) Simulation clock

The simulation must keep track of the current simulation time, in whatever measurement units are suitable for the system being modeled. In discrete-event simulations, as opposed to real-time, time "hops" because events are instantaneous: the clock skips to the next event start time as the simulation proceeds.

(4) Control components

Control components include: (i) an initialization routine, which initializes the model at time 0; (ii) a timing routine, which determines the next event time, type; advance clock; (iii) event routines, which carry out logic for each event type; (iv) library routines, which are utility routines to generate random variants, etc.; (vi) a report generator to summarize and report results at the end; (vii) the main program, which ties routines together and executes them in right order.

(5) Random-number generators

The simulation needs to generate random variables of various kinds, depending on the system model. This is accomplished by one or more pseudorandom-number generators. The use of pseudorandom numbers as opposed to true random numbers is a benefit should a simulation need a rerun with exactly the same behavior.

A simulation project is composed of experiments, each consisting of one or more replications (trials). A replication is a simulation that uses the experiment's model logic and data but a different set of random numbers, and so produces different statistical results which can then be analyzed across a set of replications. A simulation project thus involves initializing the experimental model, running it until a run-ending condition is met, and reporting results. Figure 19.11 describes such a project.

(2) Continuous simulation software

Continuous simulation classifies simulations either for a continuous process or for a continuous system, which are usually described by sets of ordinary or partial differential equations, possibly coupled with algebraic ones. Continuous simulation introduces partly symbolical and partly numerical algorithms based on the mechanisms that dynamically govern the process or system.

Modern environments relieve the occasional user from having to understand how the simulation actually works. Once the mathematics has been formulated, the modeling and simulation environment compiles and simulates it, and curves of result trajectories appear magically on the user's screen. Yet, magic has a tendency to fail, and it is then that the user must understand what went wrong, and why the model could not be simulated as expected.

Continuous simulation is a highly software-oriented task, mostly based on MATLAB, a process or system simulator. Advanced Continuous Simulation Language (ACSL) is another language designed for modeling and evaluating performance by time-dependent, nonlinear differential equations. It is a dialect of the Continuous System Simulation Language (CSSL), originally designed by the Simulations Council Inc. (SCI) in 1967 in an attempt to unify continuous simulation subjects.

Continuous simulation is often combined with discrete simulation as discrete-continuous simulation. In this approach, continuous variables are described by differential equations; and discrete events can occur that affect the continuously changing variables. Some discrete-event simulation software will also do combined discrete-continuous simulation.

(3) Network simulation software

Network simulation software is designed to model the potential behaviors of computer networks. These applications allow network engineers to simulate scenarios without a testbed of networked computers, network routers, and other, potentially expensive, hardware devices. Most products allow designers to experiment with nodes such as bridges, hubs, optical cross-connections and media access units (MAUs).

Such software is also used to model scenarios such as a sudden increase in network traffic or the effects of a denial-of-service attack. Some enable experimentation with various data layers, network layers, and routing and transportation protocols in networks consisting of point-to-point, Ethernet, and other standard-segment types.

(a) Network simulation principles

Simulation of networks can be a difficult task. For example, if congestion is high, then estimation of the average occupancy is challenging because of high variance. To estimate the likelihood of a buffer overflow in a network, the time required for an accurate answer can be fantastically large.

Most network simulations use discrete-event simulation, in which a list of pending events is stored, and those events are processed in order, with some events triggering future ones, such as the arrival of a packet at one node triggering its arrival at a downstream node.

Specialized techniques such as control variates and importance sampling have been developed to speed up simulation of data transmission in networks. Some problems, notably those relying on queuing theory, are well suited to Markov chain simulations, in which no list of future events is maintained and the simulation consists of transiting between different system states in a memory-less fashion. Markov chain simulation is typically faster, but less accurate and flexible, than detailed discrete-event simulation. Some simulations are cyclic-based, which are faster than event-based simulations.

(b) Network simulators

Often, network simulators are used to test new or changed protocols in a controlled environment. A network simulator is a program that imitates the working of a computer network, where the network is typically modeled with devices, traffic, etc., so its performance can be analysed. Typically, users can then customize the simulator to fulfill their specific analysis needs.

Most commercial simulators are GUI-driven, while some require input scripts or commands (network parameters). These parameters describe the state of the network (node placement, existing links) and the events (data transmissions, link failures, etc.). An important output of simulations is the

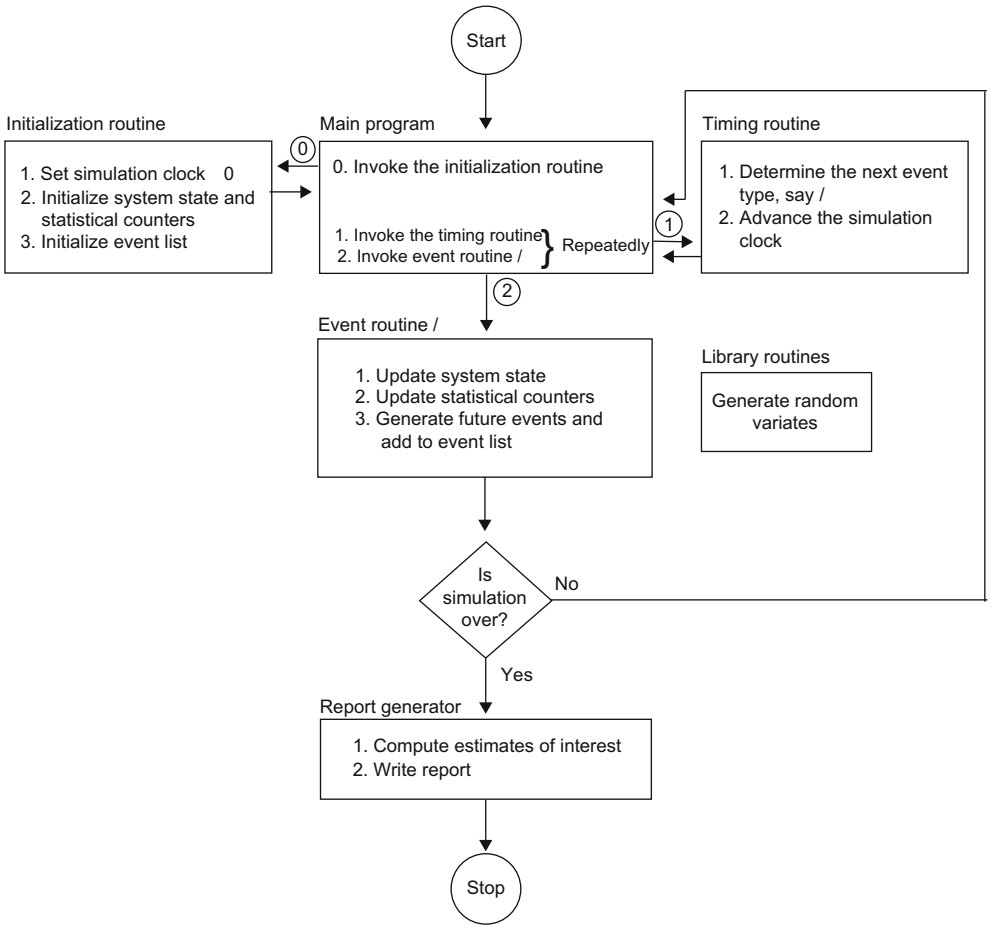


FIGURE 19.11

A discrete-event simulation project using an experiment model.

trace files, which document every event that occurred in the simulation and are used for analysis. Certain simulators have the added functionality of capturing this type of data directly from a functioning production environment, at various times of the day, week, or month, in order to reflect average, worst-case, and best-case conditions. Network simulators can also provide other tools to facilitate visual analysis of trends and potential troublespots.

(c) Network protocol simulation software

Network simulation software supports protocols such as ATM, Ethernet, IPv4, IPv6, token bus, and token ring. Asynchronous transfer mode (ATM) is a high-speed technology that uses fiber-optic or CAT-5 copper cables. Ethernet is a local-area network (LAN) protocol that serves as the basis for the IEEE 802.3 standard, and which specifies the physical and lower software layers. IPv4 and IPv6 are the

fourth and sixth versions, respectively, of Internet Protocol (IP), a technology for transmitting data across a packet-switched network. Token bus features a bus topology and uses a virtual token-passing mechanism for regulating network traffic. With token ring networks, all devices are connected in a ring or star. Network simulation software for other network protocols is also available.

(4) Electronic circuit simulation softwares

Software for electronic circuit simulation is used for designing circuits, schematics, wiring diagrams, and embedded systems (including firmware). Specialized software applications are used to draw schematics for printed circuit boards (PCBs), electronic or mechanical components, the design of microcontrollers and microprocessors, and systems for computer-aided manufacturing (CAM).

Electronic design services perform several steps to design a circuit. First, they review the customer's requirements and produce a technical report. Next, they draft a schematic or circuit diagram to meet the project's specifications. After calculating the component values, they perform simulations to verify the correctness and accuracy of the circuit design. Unless changes are required, a breadboard or other prototype version is then built. Electronic design services also select parts and materials, recommend a production method, and present their work to the customer. A number of prototypes are tested or type-tested to ensure compliance with the project's requirements.

Electronic design services differ in terms of capabilities. Some companies specialize in the design of digital electronics and embedded systems, others in PCB layouts for analog electronics.

(a) Computer-based circuit simulators

Computer-based circuit simulation is a two-step process. The first step must construct the actual circuit diagram using wires and electronics components (i.e. resistor, capacitors, inductor, diodes, integrated circuits, etc…). The second step varies the inputs to the circuit and to see how it affects operation and outputs by calculating ideal theoretical behavior from Kirchhoff's laws (see Wikipedia).

The industry standard analog circuit simulator is SPICE (Simulation Program with Integrated Circuit Emphasis), which was originally developed at the University of Berkeley during the 1970s. SPICE (v2G.6) is the basis for many commercial computer software programs, which provide the GUI (graphical user interface), but use the SPICE (or WinSPICE) simulation engine to perform all the circuit calculations. Figure 19.12(B) is an illustration of SPICE's GUI window.

SPICE does not simulate the electromagnetic fields in a circuit, since these depend explicitly on its layout. Results are trust worthy to the low MHz range, but should be treated with suspicion for higher frequencies.

(b) Computer-based circuit layout editor

In a professional setting, the layout of an electronic circuit determines its compactness, ease of use (and debugging), cost, longevity, and performance (especially at high frequencies). A number of programs exist to help to design them. In fact, most electronics engineers will design an abstract circuit with a circuit simulator and then use a software package to lay out the actual circuit on a PCB. This design is then turned into an industry standard Gerber file, which is then sent to a PCB production company which will assemble a prototype and test it. Once it is tested, the entire production is usually contracted out to a third company. The use of professionally made PCBs is relatively common, since it generally results in a reproducible circuit, which is likely to work better at high frequencies than one assembled from wires or prototyping boards.

19.3.2 Industrial simulation tools

There are numerous industrial simulation tools for various applications, of which four are briefly introduced in this section: MATLAB, LabVIEW, SIMULINK and ModelSim.

(1) MATLAB

MATLAB is an integrated and technical computing environment that combines numeric computation, advanced graphics and visualization, and a high-level programming language. It gives an interactive system whose basic data element is an array that does not require dimensioning, which allows it to solve technical computing problems, especially those with matrix and vector formulations very quickly.

MATLAB has evolved over a period of years with contributions from many users. In university environments, it is the standard instructional tool for mathematics, engineering, and science. In industry, it is the tool of choice for high-productivity research, development, and analysis, used in a variety of application areas including signal and image processing, control system design, financial engineering, and medical research. Typical facilities provided by MATLAB include mathematical operations; numerical computation; algorithm development; data acquisition; modeling, simulation, and prototyping; data analysis, exploration, and visualization; scientific and engineering graphics; and application development including graphical user interface (GUI) building.

Several software toolkits for developing specialized applications are available from technical vendors. All the toolkits integrate MATLAB seamlessly. Refer to related websites for more information.

(2) LabVIEW

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a graphical programming language that uses icons instead of lines of text to create applications. In contrast to text-based programming languages, where instructions determine the order of program execution, LabVIEW uses dataflow programming, where the flow of data through the nodes on the block diagram determines the execution order of the virtual instruments and functions. Virtual instruments are LabVIEW programs that imitate physical instruments.

In LabVIEW, a user interface can be built by using a set of tools and objects. The user interface is known as the front panel. Once an interface has been built successfully, code can be added by using graphical representations of functions to control the front panel objects. This graphical source code is known as G code or block diagram code. The block diagram contains this code. In some ways, the block diagram resembles a flowchart.

Several add-on software toolkits for developing specialized applications can be purchased; all integrating LabVIEW seamlessly. Refer to the National Instruments website at www.ni.com\toolkits for more information about these tools.

(3) SIMULINK

SIMULINK is an interactive tool for modeling, simulating, and analyzing dynamic systems. It integrates seamlessly with MATLAB, providing immediate access to an extensive range of analysis and design tools. It supports linear and nonlinear systems, continuous-time and sampled-time systems, and hybrid systems. With SIMULINK, simulations are interactive, so that parameters can be changed

and the effect observed immediately. It moves beyond idealized linear models to explore more realistic nonlinear models, and has instant access to all of the analysis tools in MATLAB to take the results, analyze and visualize them.

For modeling, SIMULINK provides a graphical user interface for building models in block diagrams, using click-and-drag mouse operations. With this interface, models can be drawn as they would be with pencil and paper (or as most textbooks depict them). This is much better than packages that require you to formulate differential and difference equations in a language or program.

The model can then be simulated, choosing integration methods, either from the SIMULINK menus or by entering commands in the MATLAB window. The menus are particularly convenient for interactive work, while the command-line approach is very useful for batch simulations (e.g., if you want to sweep a parameter across a range of values). Using scopes and other display blocks, results can be watched while the simulation is running. In addition, parameters can be changed and the effect

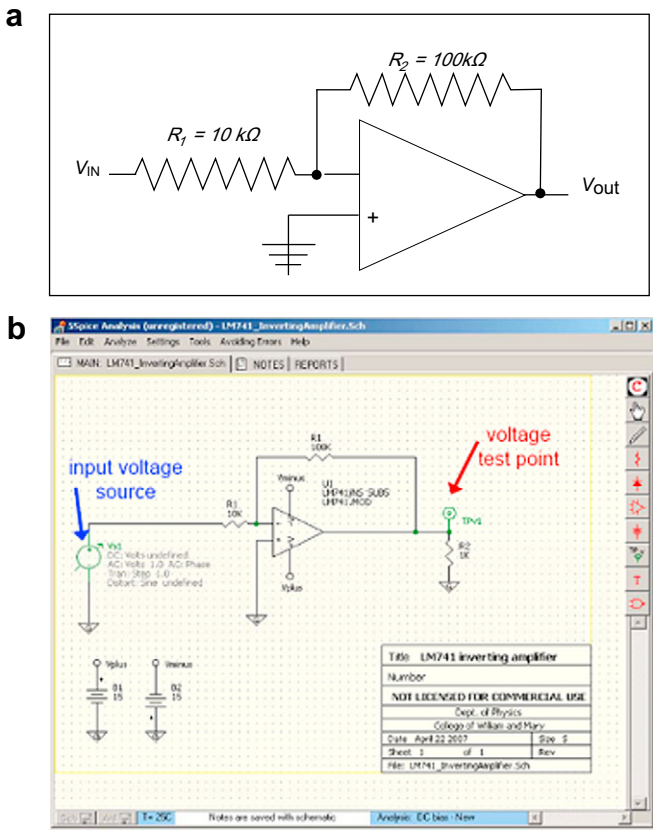


FIGURE 19.12

An example using the electronic circuit simulation software SPICE version v1.22 to design application-specific integrated circuits. (A) The target circuit of this design, which is an amplifier of gain = 10; (B) one of SPICE's GUI windows, displaying the final circuit diagram schematic of this design.

observed, for what-if exploration. The simulation results can be put in the MATLAB workspace for post-processing and visualization.

(4) ModelSim

ModelSim is the industry-leading, Windows-based simulator for very high-speed integrated circuits (VHDL), Verilog, or mixed-language simulation environments. ModelSim offers VHDL, Verilog, or mixed-language simulation. Coupled with the most popular HDL debugging capabilities in the industry, ModelSim is known for delivering high performance, ease-of-use, and outstanding product support.

It delivers a unique combination of native compiled code architecture and outstanding simulation performance. An easy-to-use graphical user interface enables the user to identify and debug problems quickly, aided by dynamically updated windows. For example, selecting a design region in the structured window automatically updates the Source, Signals, Process, and Variables windows. Once a problem is found, you can edit, recompile, and resimulate without leaving the simulator. ModelSim fully supports the VHDL and Verilog language standards. You can simulate behavioral and gate-level code separately or simultaneously. ModelSim also supports all application-specific integrated circuit (ASIC) and field-programmable gate array (FPGA) libraries, ensuring accurate timing simulations.

Problems

1. What is an open loop control system? What is a closed loop control system?
2. There are several possibilities for classifying process identification methods: (1) passive and active identification methods; (2) deterministic and stochastic identification methods; (3) single and iterative identification methods. Please briefly explain these classes of process identification methods.
3. A house (or room) heating system consists of the boiler, one or more radiators, and thermostat (controller). First design a closed loop house (or room) heating system. For the designed heating system, use the black box modeling approach to generate a linear equation model between the gas inputs and the hot air outputs of the boiler for the thermostat (controller) to control the house temperature in terms of the target and real temperatures in the house (or room). Verify whether or not this linear model is a parametric or non parametric model. Then write the model identification formulas by linear least squares if it is a non parametric model; or the state space method if it is a parametric model.
4. A house (or room) heating system consists of the boiler, one or more radiators, and thermostat (controller). First design a closed loop house (or room) heating system. For the designed heating system, use the black box modeling approach to generate a nonlinear equation (a quadratic polynomial) model between the gas inputs and the hot air outputs of the boiler for the thermostat (controller) to control the house temperature in terms of the target and real temperatures in the house (or room). Verify whether or not this nonlinear model is a parametric or non parametric model. Then write the model identification formulas by the prediction error method if it is a non parametric model; or the state space method if it is a parametric model.
5. Change problem 3 from a closed loop into an open loop system, and then complete it.
6. Change problem 4 from a closed loop into an open loop system, and then complete it.
7. An automobile speed control system consists of the engine, accelerator pedal (controller), and wheels. Find the relation curves of the engine's gasoline inputs to its outputs (only crankshaft's speeds) for a car or lorry. Then check whether this relation curve is linear or nonlinear and write the accelerator pedal (controller) control model equation.
8. Continuing problem 7, add the crankshaft torques (or loadings of the automobile) as another variable. Then write the accelerator pedal (controller) control model equation(s) again.
9. What is your understanding of "process modeling and simulation"? What is your understanding of "system modeling and simulation"?
10. What is your understanding of the linkage and dependence between model, simulation and control?

11. List all the model types in the manufacturing processes described in subsection 19.2.1, and then discuss the modeling complexity of industrial control.
12. What are linear and nonlinear PID (proportional integral derivative) controllers? Then explain why PID controllers do not perform well when applied to systems with significant time delays.
13. Please explain why the "Inverse of process model" in Figure 19.6 is almost impossible to find for many types of industrial processes.
14. Please explain the working principle of the adaptive controller in Figure 19.7, especially how can this controller be defined as an adaptive controller, Please explain the working principle of the adaptive predictive controller in Figure 19.9, especially how can this controller be defined as an adaptive predictive controller. (Note: Figure 19.8 may help you with this problem.)
15. What is the meta modeling method?
16. Suppose you are appointed as an engineer responsible for designing a simulation model for an urban traffic control system which manipulates the traffic lights according to vehicle flows and time schedules at crossroads. Please first give all events, entity lists, and control routines of your simulation model; please then work out all the continuous processes of your simulation model; finally, complete your design for a discrete continuous simulation model for this urban traffic control system.
17. Based on Figure 19.11, which is the flow chart of a discrete event simulation project, develop a computer program package in whatever programming language you like for the discrete event part of your simulation model of an urban traffic control system.
18. Advanced Continuous Simulation Language (ACSL) is another computer language designed for modeling and evaluating the performance of continuous processes and systems described by time dependent, nonlinear differential equations. Please study ACSL and list all the differences from a computer programming language, say C++.
19. What are Kirchhoff's circuit laws?
20. Trace files are important outputs of simulations. Please say where and how you would get the trace files from simulations.
21. Find some MATLAB tutorials from the Internet to learn more details of this simulation tool.
22. From National Instruments' web site, www.ni.com\toolkits, you can find the LabVIEW manual and fundamental documents. Please read these documents to learn its basics first, including "Virtual Instruments" and the LabVIEW environment; then start building the front panel templates.
23. How can you make combinations of the simulation software classes given in subsection 19.3.1 of this textbook in order to design a simulator for an industrial programmable controller, CNC or PID controller?

Further Reading

William S. Levine (Ed.). The Control Handbook. CRC Press. 2000.
Dobrivoje Popovic, Vijay P. Bhatkar. Distributed Computer Control for Industrial Automation. CRC Press. 1990.
Philip Thomas. Simulation of Industrial Processes for Engineers. Butterworth Heinemann. 1999.
Robert W. Lewis. Programming Industrial Control Systems Using IEC 1131 3 Revised Edition. The Institution of Electrical Engineers, London. 1998.
J. A. Rossiter. Model based Predictive Control: A Practical Approach. CRC Press. 2003.
Christian Schmid. Open loop and Closed loop. 2005 05 09. http://www.atp.ruhr uni bochum.de/rt1/syscontrol/node4.html. Accessed: April 2008.
Su Whan Sung et al. Process Identification and PID Control. PJohn Wiley & Sons. 2009.
Yucai Zhu. Multivariable System Identification for Process Control. Elsevier Science. 2001.
Kaddour Najim, Enso Ikonen. Advanced Process Identification & Control. CRC Press. 2001.
Jan Mikles, Miroslav Fikar. Process Modelling, Identification, and Control. Springer. 2007.

Marcos R. Vescovi et al. Modeling and simulation of a complex industrial process. http://portal.acm.org/citation.cfm?id=630235. Accessed: November 2009.

Wikipedia (http://en.wikipedia.org). Computer simulation. http://en.wikipedia.org/wiki/Computer simulation. Accessed: November 2009.

Wikipedia (http://en.wikipedia.org). Discrete event simulation. http://en.wikipedia.org/wiki/Discrete event simulation. Accessed: November 2009.

Wikipedia (http://en.wikipedia.org). Kirchhoff's circuit laws. http://en.wikipedia.org/wiki/Kirchhoff%27s circuit laws. Accessed: November 2009.

SteelLink (http://steellinks.com). Automation and control: process automation, control and simulation. http://steellinks.com/pages/index.html. Accessed: November 2009.

R. Jacob Baker et al. CMOS Circuit Design, Layout, and Simulation. Wiley IEEE Press. 1997.

Francois E. Cellier, Ernesto Kofman. Continuous System Simulation. Springer. 2006.

Ultra cold AMO Physics Laboratory, College of William & Mary. Electronic Circuit Simulation and Layout Software, Chapter 12. 2008.

Thomas J. Schriber, Daniel T. Brunner. Inside Discrete event Simulation Software. Proceedings of the 1997 Winter Simulation Conference, Honolulu. pp.14 23.

GlobalSpec (www.globalspec.com). Network simulation software. http://www.globalspec.com/LearnMore/Network Simulation Software. Accessed: November 2009.

GlobalSpec (www.globalspec.com). About software development services. http://www.globalspec.com/LearnMore/Software Development Services. Accessed: November 2009.

MathWorks (www.mathworks.com). MATLAB. http://events.unisfair.com/index.jsp/matlab virtual. Accessed: November 2009.

MathWorks (www.mathworks.com). SimuLINK. http://www.mathworks.com/products/slhdlcoder. Accessed: November 2009.

National Instruments (www.ni.com). LabVIEW fundamentals. http://www.fzu.cz/texts/labview/LV Fundamentals.pdf. Accessed: November 2009.

National Instruments (www.ni.com). LabVIEW manual. ftp://ftp.ni.com/pub/devzone/tut/manual.pdf. Accessed: November 2009.

Model (www.model.com). ModelSim: http://www.model.com/content/modelsim. Accessed: November2009.

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z