Youn-Long Steve Lin
Editor

Essential Issues in SOC Design

Designing Complex Systems-on-Chip

Springer

Essential Issues in SOC Design

Essential Issues in SOC Design

Designing Complex Systems-on-Chip

Edited by

Youn-Long Steve Lin

National Tsing Hua University, Taiwan

Springer

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN-10 1-4020-5351-7 (HB)
ISBN-13 978-1-4020-5351-1 (HB)
ISBN-10 1-4020-5352-5 (e-book)
ISBN-13 978-1-4020-5352-8 (e-book)

Published by Springer,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

www.springer.com

Printed on acid-free paper

Table of Contents

Contributing Authors

Tian-Sheuan Chang, National Chiao-Tung University
Yao-Wen Chang, National Taiwan University
Chien-Liang Chen, Global UniChip Corp.
Ching-Yeh Chen, National Taiwan University
Huang-Yu Chen, National Taiwan University
Liang-Gee Chen, National Taiwan University
Tung-Chieh Chen, National Taiwan University
Edward T.-H Chu, National Tsing Hua University
Moo-Kyoung Chung, Korea Advanced Institute of Science and Technology
Chih-Tsun Huang, National Tsing Hua University
Tai-Yi Huang, National Tsing Hua University
Tohru Ishihara, Kyushu University
Ando Ki, Dynalith Systems Co., Ltd.
Young-Il Kim, Dynalith Systems Co., Ltd.
Chong-Min Kyung, Korea Advanced Institute of Science and Technology
Jae-Gon Lee, Korea Advanced Institute of Technology
Kun-Bin Lee, Mediatek Inc.
Chung-Jr Lian, National Taiwan University
Jiing-Yuang Lin, Global UniChip Corp.
Youn-Long Lin, National Tsing Hua University
Ko-Yun Liu, National Tsing Hua University
Masanori Muroyama, Kyushu University
Shiao-Li Tsao, National Chiao-Tung University
Cheng-Wen Wu, National Tsing Hua University
Le-Chun Wu, National Taiwan University
Wooseung Yang, Dynalith Systems Co., Ltd.
Hiroto Yasuura, Kyushu University

Chapter 1

ESSENTIAL ISSUES IN SYSTEM-ON-A-CHIP DESIGN

Youn-Long Lin
Department of Computer Science, National Tsing Hua University, Hsin-Chu, TAIWAN

Abstract: Due to advance in semiconductor manufacturing technology, integration of whole electronics system on a single chip is feasible. Starting with baseline CMOS logic, semiconductor wafer manufacturers have gradually added to their portfolio embedded memory (SRAM, OPT, Flash), mixed signal devices, RF devices, and even MEMS. Because it offers many advantages over traditional multiple-chip solutions, system-on-a-chip (SOC) has drawn great attention from both academia and industry. We expect an SOC solution to be smaller, less expense, more energy efficient, more reliable, etc. However, designing an SOC for successful mass production is much more complicated than that of traditional simpler logic, memory, or analog chips. This chapter outlines some important issues that face an SOC design team and give brief introduction to each chapter of this book

Keywords: System-On-a-Chip, SOC Design Foundry, Multimedia SOC

1. INTRODUCTION

Since the integrated circuit was invented in 1958, the number of devices that can be massively-produced on a chip has been increased following the Moore's Law, that is, the number of transistors on a chip doubles every 18 months or so. In the old days when a chip contains only smaller number of transistors, an electronics system consists of a large number of chips housed in many printed-circuit boards which in turn are put into a cabinet. Hence, we can call them system-on-boards. Nowadays, semiconductor manufacturing process can give us a billion-transistor chip for a few US dollars. This makes possible applications that were previously either impossible or unaffordable.

Ever increasing computational demand from the application side and very deep submicron semiconductor processing from the technology side together make system-on-chip (SOC) reality and necessary.

To design an SOC for successful mass production, we have to coupe with many technical and management issues. Here we focus on the technical aspect.

Let's begin with what constitute an SOC. Like a typical electronics system, an SOC consists of processing elements, I/O devices, storage elements and interconnection structure linking all of them together.

Processing elements could be processors that run embedded software or functional-specific hardware accelerators. There are two popular processor categories: microprocessor for control and management function, and digital signal processor (DSP) for signal processing-specific function. Recently, there are academic and industrial efforts in the so called Application-Specific Instruction set Processor (ASIP), which allows instruction set extension by the users according to the target applications. It is quite common that multiple types and multiple instances of processors are used in a single SOC project. For example, in the TI-OMAP SOC, an ARM microprocessor and a TI DSP core co-exist.

When a software approach cannot deliver adequate performance for an application, we turn to dedicated hardware blocks. Typical accelerators include JPEG image Codec, MPEG-4 Video Codec, Viterbi Decoder, Turbo Code Decoder, AES Encryption/Decryption engines, etc.

To communicate with outside, an SOC usually consists of many types of standard I/O devices. Commonly found I/O IPs include Ethernet MAC and Phy, USB1.1/2.0 Device Controller and Phy, other high-speed serial links such as LVDS (Low Voltage Differential Signaling), Audio/Video Output, Memory Controller, etc.

Both internal and external memories are important to SOC. A typical SOC utilizes hundreds of internal memory blocks. They may be SRAM, ROM, Flash or OTP (One-Time Programmable). Their configurations in terms of number of words, word length, number of read/write access ports, and access speed are all tailor made to fit the applications.

When a memory block is too large to be effectively made on chip, we usually put it off-chip and integrate it with the SOC using a system-in-package (SiP) solution. Commonly used external memory includes DRAM and Non-Volatile Flash Memory. Very often we will find an SiP packed in an SDRAM or DDR-II die. Therefore, the SOC has to include memory controller for SDRAM, DDR, SD Card, MMC Card, Compact Flash, etc.

With all components available, we need a communication structure to put them all together. This is called on-chip-communication or on-chip-bus. Just like the PCI-bus of the PC system allows easy plug-and-play of memory cards, graphics cards, etc, the SOC community has proposed several on-chip-bus standards. One of the most popular bus architecture is the Advanced Microprocessor Bus Architecture (AMBA) by ARM.

As on-chip communication traffic exponentially increases and deep submicron effect makes transferring signals difficult in single cycle, researchers have proposed Network-on-Chip (NoC) communication architecture to cope with the problem. An NoC brings the computer networking technology (i.e., packet routing) to the SOC in order to simplify the design and management of communication among IPs in an SOC.

To design a complex SOC, we have to deal with the following essential issues: (1) Availability of components, (2) System integration and verification, and (3) Physical implementation.

Components used in an SOC are also called silicon intellectual property (IP). The SOC development team has to decide on which IP to use or design. There are many IP vendors each serving some segments of the IP market. For processor IP, software compatibility must be taken into account. For memory and I/O IPs, whether they have mass-production record is the main concern.

In case there are not suitable, ready to use IPs, we have to modify an existing one or develop a new one. Longer turn-around time, higher risk, and greater resource needs must be taken into account.

After we put all components together into a system, we have to verify its functional and timing correctness. In the old days when the chip was small, we usually relied on simulation tools for verification. However, for complex SOC which have high gate count and longer simulation pattern, simulation alone cannot give us sufficient confidence level. Moreover, as processors are integrated, we have to perform software/hardware co-verification down to cycle-accurate level. To cope with this challenge, emulation based prototyping is needed.

Physical implementation of SOC is also more difficult than that of traditional ASIC. Complex SOCs are usually targeted towards advanced nanometer technology (90nm and below). As feature size shrinks, process variation becomes relatively significant. Variation-aware analysis and optimization of timing and power consumption must be introduced into the implementation flow. Design for manufacturability consideration becomes a must.

In the case where system-in-package is chosen, chip and package co-design is inevitable.

After an SOC is tape-out, the design team should work closely with the testing team and the processing engineers to enhance the yield.

2. BOOK OVERVIEW

This book brings together experts from different research areas to present their knowledge in various topics related to SOC design. We hope that they have pointed to possible solutions and research directions for those who are either designing SOCs or are considering entering the field.

Chapter 2 describes "An SOC Controller for Digital Still Camera." This is a real industrial case. The authors present their experience in defining specification with system house, taking IP from third parties, integrating the SOC and, finally, ramping up for mass production of millions of units.

Chapter 3 presents "Multimedia IP Development – Image and Video Codec." The authors describe their academic experience in developing JPEG, JPEG2000 and MPEG4 codec IPs that find their ways into industrial applications.

Chapter 4 deals with "SOC Memory System Design." Memory will account for majority of silicon area in most SOCs. There is trade-off between memory usage and memory traffic. Careful algorithm and architecture designs will gain significantly in terms of area, performance and power consumption.

Chapter 5 describes "Embedded Software." Contemporary SOCs all contain one or more microprocessors and DSPs. Both system software and application software are important. Unlike traditional PC-based software, embedded software must have small foot-print and consume less power. Moreover, their interaction with hardware devices and hardware accelerators is more closely coupled.

Chapter 6 presents "Energy Management Techniques for SOC Design." Since most SOC solutions are for portable devices, which have limited battery life, power efficiency is a major concern. It is well known that the power of a circuit is linearly proportional to the frequency and quadratic proportional to the supply voltage. Depending on the characteristics of applications, we can run circuits at various clock rates to just meet the deadline. Consequently, slow circuit needs only small supply voltage. Dynamically scheduling the frequency and voltage will result in significant energy saving.

Chapter 7 describes "SOC Prototyping and Verification." It takes a long time and huge costs to get a complex SOC manufactured. We cannot afford to make any mistakes during the design process. Complete verification of functionality and timing is a must for any SOC project. To speed up the verification process, prototyping is a popular approach. This chapter presents an industrial strength verification strategy.

Chapter 8 deals with "SOC Testing and Design for Testability." Testing is the key to a high quality product. In a complex SOC, we should be able to test every IP in the shortest possible test application time. Therefore, test integration and scheduling are important issues. Moreover, design for testability enhancement is also a common practice. For example, memory BIST has to be inserted into every memory macros.

Chapter 9 describes "Physical Design for SOC." In the nanometer semiconductor manufacturing process, chip complexity and process variation together make physical implementation challenging. High complexity calls for hierarchical divide-and-conquer approach, while process variation calls for statistical-based analysis and optimization. A chip should be laid out such that it is manufacturable (i.e., high yield). Therefore, physical design should be aware of a mask making process and the manufacturing process.

Chapter 2

A SOC CONTROLLER FOR DIGITAL STILL CAMERA

Jiing-Yuang Lin,* Chien-Liang Chen,* and Youn-Long Lin**
**Global UniChip Corp., Hsin-Chu, TAIWAN*

*** Department of Computer Science, National Tsing Hua University, Hsin-Chu, TAIWAN*

Abstract: We present our experience of designing a single-chip multimedia SOC for advanced digital still camera from specification all the way to mass production. The process involves collaboration with camera system designer, IP vendors, EDA vendors, silicon wafer foundry, package & testing houses, and camera maker. We also co-work with academic research groups to develop a JPEG codec IP and memory BIST and SOC testing methodology. In this presentation, we cover the problems encountered, our solutions, and lessons learned. This case study shows the feasibility of expanding semiconductor wafer foundry service to electronics manufacturing service (EMS) providers who in general have very limited IC design capability/experience. We also point out possible directions for future research

Keywords: System-On-a-Chip, SOC Design Foundry, Multimedia SOC, Silicon Intellectual Property, Design for Manufacturability

1. INTRODUCTION

Ever increasing computational demand from the application side and very deep submicron semiconductor processing from the technology side together make system-on-chip (SOC) reality and necessary. Makers of such electronics systems as PDA, cellular phone handsets, digital still camera, portal music player, etc., need Application-Specific Integrated Circuits (ASIC) solutions in order to differentiate themselves from the competition, to increase product value, and to reduce cost. On the other hand, semiconductor wafer foundry has to expand its service scope from wafer manufacturing to mask tooling, cell & I/O library, memory compiler, and up to silicon intellectual properties (IP) such as Phase-Lock Loop (PLL), Digital-to-Analog Converter (DAC), and Analog-to-Digital Converter (ADC). Therefore, there is a need to bridge the gap between electronic system houses and wafer foundry. We call such company SOC design service provider.

An SOC design service provider takes as its inputs from the electronics system house a specification or partially-designed prototype and delivers to its customer layout database in GDSII format ready for manufacturing as depicted in Figure 1. It is also called a fabless ASIC vendor if packaged and tested chips are delivered instead. Close collaboration is needed among all parties in order to successfully bring a competitive product to the market in time.

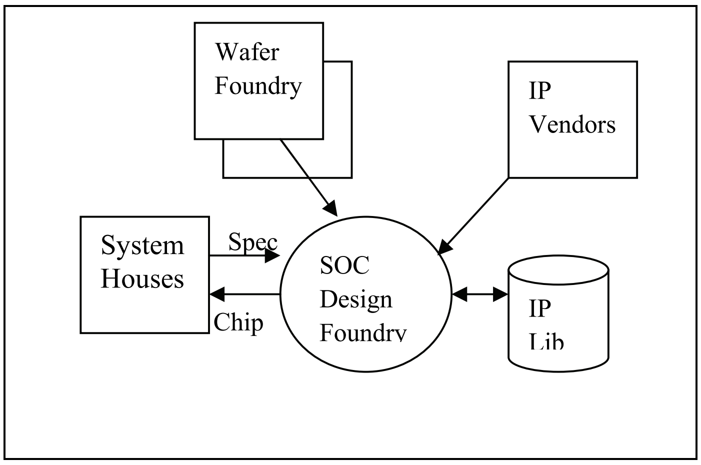


Figure 1. SOC design foundry

System houses are also called Electronics Manufacturing Service (EMS) as they do design and manufacture but they do not sell products under their own brands. Instead, their customers are those brand-name companies. Presently, almost all IT products including PC, Notebook, cellular phone, PDA, digital camera, music player, etc., are all operated under this business model. EMS usually does not have IC design capability. Instead, they buy chips from IC design houses and differentiate from one another in system board level design and software. As chip integration level increase, the room for differentiation in the board level shrinks. Therefore, it is natural for them to search for their own chip solutions (ASIC). As EMS usually command huge volume in the order of tens of millions units per year, it is reasonable and economically feasible for spinning their own chips. However, chip design is not their core business. Hence, partnership with a chip design service provider becomes essential to an EMS's competitiveness.

In the semiconductor manufacturing side, the industry is divided into three segments: wafer foundry, packaging and testing houses. In the past, a semiconductor wafer foundry takes GDSII layout database from its customer and delivers manufactured wafers. As technology advances and design complexity grows, more and more customers cannot afford expensive infrastructure and investment required to produce GDSII in house. Wafer foundry can expand its reach of service to those who cannot submit GDSII by teaming up with an SOC design service provider.

For package and testing houses, it is beneficial to co-work with a design service provider too. It is already well known that design for testability is commonly accepted practice. Presently, heterogeneous integration of logic, memory, and radio frequency (RF) devices, makes testing and diagnosis more complicated. Therefore, it is essential to involve testing houses in an SOC design project. As package technology advances, substrate design, pin-to-pad routing, thermal aware package design, layout-package co-design all become very important. Moreover, system-in-a-package (SiP) is gaining momentum. It is very common to pack an SOC together with a DRAM and/or a Flash in a same package. Therefore, cooperation between SOC design service foundry and packaging service providers is also essential.

One of the promising approaches to cope with high design complexity is reusing existing design from previous projects or external sources. Such reusable object is called silicon intellectual property (IP). There are many IP vendors specializing in microprocessor (i.e., ARM, MIPS, Tensilica), digital signal processor (DSP), embedded memory (SRAM, 1T-RAM, ROM compilers), standard interface (USB, Ethernet), analog blocks (PLL, ADC, DAC), accelerators (JPEG, MPEG), etc. We have also seen organization that promotes inter-operability of IPs (e.g., OCP-IP). Usually, it is a tedious process for an SOC project manager to put together all appropriate IPs because there are many uncertainty and ambiguity in diverse IP from diverse sources. It is beneficial to have a one-stop-shopping service such as an SOC design service foundry, which has multiple experiences with various IP. Therefore, productivity is increased and risk reduced.

Digital still camera (DSC) is one of the fastest growing consumer electronics products over the past few years. Due to the success of the JPEG image compression standard, advance in CMOS image sensing and availability of high capacity yet low cost flash memory cards, DSC has virtually taken over the traditional film-based camera in just a few years. Moreover, DSC also penetrates quickly into cellular phone sets, which have become the convergent target of PDA, MP3 audio player, etc. Ever increasing picture resolution and advanced features such as video clip recording requires ultra low power and small form factor integration of all needed functionality. Therefore, an SOC solution is very attractive to the camera makers.

We describe our experience with designing an SOC for DSC controller applications including IP preparation, system integration and verification, chip implementation, manufacturing, failure analysis and yield enhancement during million-units mass production. In Section 2, we first give the chip specification defined by the camera system maker. Then, we list all the intellectual properties (IP) used and difficulty encountered. The integration and verification of the whole system in a chip is then described. Section 3 presents our chip implementation flow from RTL synthesis down to GDSII layout ready for manufacturing. Then, we describe mass-production-related issues including yield ramp-up and failure analysis. Section 4 describes recent development based on the presented project. Finally, we summarize this chapter in Section 5.

2. A DIGITAL STILL CAMERA SOC

Our objective was to design a single chip controller for 2-million-pixel and 3-million-pixel grade DSC for mass production of 3.5 million units in a span of about 18 months in year 2002 and 2003. In order to satisfy required functionality at a very aggressive cost set to help proliferating the entry-level high-resolution camera, the SOC was specified to include the following IPs:

- A microprocessor capable of both traditional 32-bit RISC and DSP functionality
- A hardwired JPEG encoding and decoding accelerator
- A hardwired custom logic for color image processing
- A USB 1.1. device controller with min-host function and its transceiver PHY
- A dual mode SD/MMC flash memory card host interface
- An SDRAM controller interface
- An LCD interface controller for view-finder
- An NTSC/PAL TV signal encoder for viewing photos on TV
- A 10-bit Video DAC for TV
- An 8-bit LCD DAC
- Two PLLs for clock sources
- 30 SRAM macros for internal buffering

The IP cores come from multiple sources for different reasons. Each of them posts different challenges to the project team. To help their development, to verify the functionality of each individual IP as well as customize some of the IP for the project, we built an SOC platform as depicted in Figure 2. In the platform, some IPs are existing ICs, some are soft cores that can be configured and programmed into the FPGA. All IPs are interconnected together with an AMBA-AHB/APB on-chip bus system. Because most of the IPs on the platform have been proven many times in previous projects, for each new SOC project we only have to concentrate on verifying newly added or customized IPs. Moreover, whole system verification is also easy due to the readiness of system-level verification bench. This platform approach greatly increase our productivity of IP development, IP qualification, and system verification.

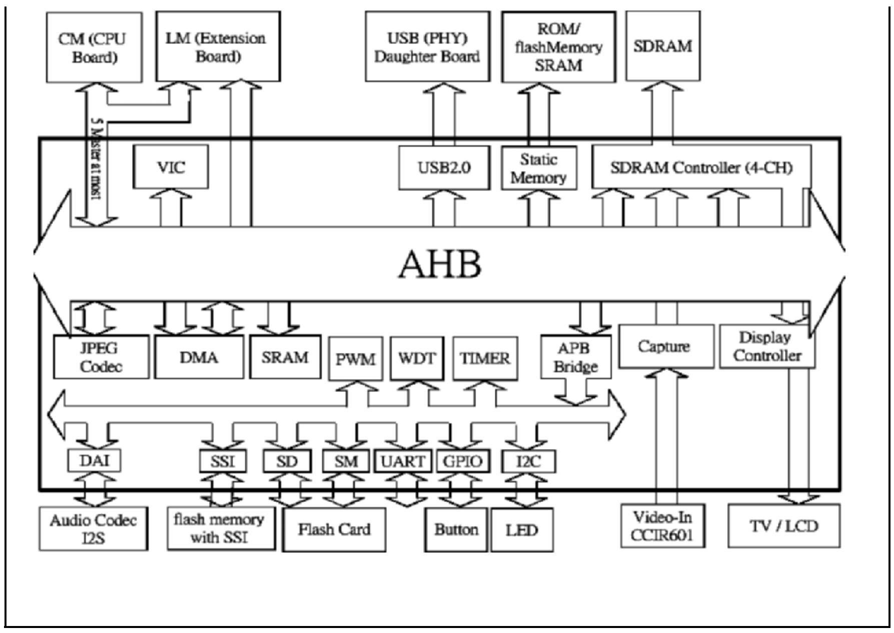


Figure 2. SOC and IP development platform

The hybrid RISC/DSP implements both a typical 32-bit RISC instruction set and a DSP-specific instruction set in a unified instruction set architecture to simplify the programming interface. It was not an IP at all. Actually it was a stand alone processor chip used in the previous generations of cameras. For software compatibility concern, we have no option but to replace it with any other IP-style microprocessor such as ARM or MIPS cores. To meet high speed requirement (133MHz @ 0.25um), we have to make it a hard core before integration with other parts of the SOC. To integrate it into the SOC, we have to collaborate with the original vendor to create synthesis, simulation and test models in addition to hardening the processor into a high-speed hard macro.

The USB1.1 device controller and the SD card (secure digital flash memory card) controller are supplied by a third party vendor. They are in

VHDL RTL instead of more locally popular Verilog. Therefore, mixed-language simulation environment has to be set up. Only FPGA prototyping was performed at the time of SOC integration. Moreover, the synthesis scripts and testbenches were less than ideal. Therefore, close intensive co-work/co-debugging was carried remotely.

To meet processing speed requirement of 3M pixels @ 0.1Sec and long battery life, the JPEG codec function has been implemented in a hardware accelerator. We collaborated with a university research laboratory. The effort we spent was in bridging the gap between university prototype and industrial strength design. Also there was discrepancy among the interpretation of the JPEG standard by the system house and the IP developers. Therefore, we added a wrapper around it as depicted in Figure 3. Extensive regressive test of more than 1,000 pictures from different origins was conducted for every change made.

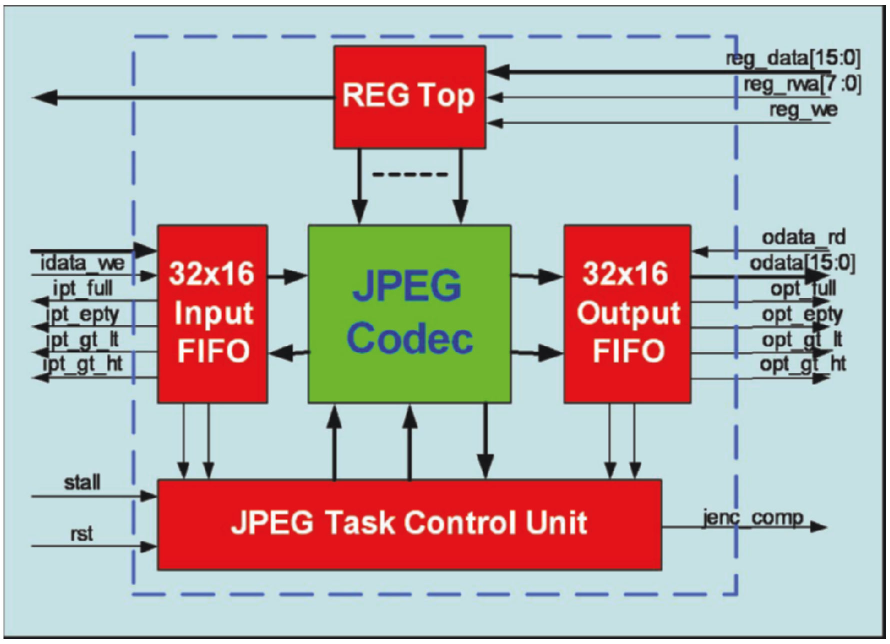


Figure 3. Wrapping a JPEG codec for the SOC platform

There is a block of custom logic for color image processing. Its function includes auto focusing, auto white-balancing, color image quality enhancement, etc. It was supplied by the camera maker in Verilog RTL.

There were more than 30 SRAM macros used in the SOC. We have jointly developed a memory BIST (Built-In Self Test) generator, again with a university laboratory. The generated BIST circuit performs testing of 100% coverage without patterns from the tester machines. Therefore, testing cost is greatly reduced during production.

After all IP models are made ready, whole system integration and verification is an even bigger challenge. We encountered the problem of in-consistent and in-sufficient testbenches. Therefore, developing testbench as the project goes is very important.

Our verification set up is a mixture of simulation and FPGA/chip co-emulation.

3. CHIP IMPLEMENTATION

Figure 4 depicts our chip implementation flow from RTL to GDSII ready for tape-out. The DSC controller consists of 240K gates excluding memory macros. After whole system verification with hybrid emulation/simulation, it was implemented in TSMC 0.25um 1P5M CMOS process and packed in TFBGA256 package. It took three months for a team of six engineers to complete the Netlist-to-GDSII implementation. During the course, there are 3 spec changes involving re-synthessi and FF modification, 10 netlist changes involving ECO of combinational logic, 3 ECO changes to fix setup/hold time violation, and 13 versions of pin assignments to simplify the substrate design.

There are 30 embedded memory macros in the controller. We use an in-house memory BIST circuit generator to insert one common BIST controller, multiple sequencers, and 30 pattern generators. The MBIST is from collaboration between us and a university research laboratory. After scan insertion, the fault coverage was 93%.

The physical design of the chip was done with timing-driven placement and routing, physical synthesis, formal verification and STA QoR check.

During chip implementation, we encountered several problems:

- During the course, there are 3 spec changes involving re-synthessi and FF modification, 10 netlist changes involving ECO of combinational logic, 3 ECO changes to fix setup/hold time violation, and 13 versions of pin assignments.
- There existed inconsistency between simulators/versions among customer, IP vendors and ourselve. The customer used PC-based Verilog/Modelsim while we used NC-Verilog. This lead to extra twist during ASIC sign-off.
- IP quality is less than ideal. We have to clean up many DRC/LVS violation in the database provided by the IP vendors.
- The USB IP was delivered in FPGA-targeted RTL. No robust synthesis

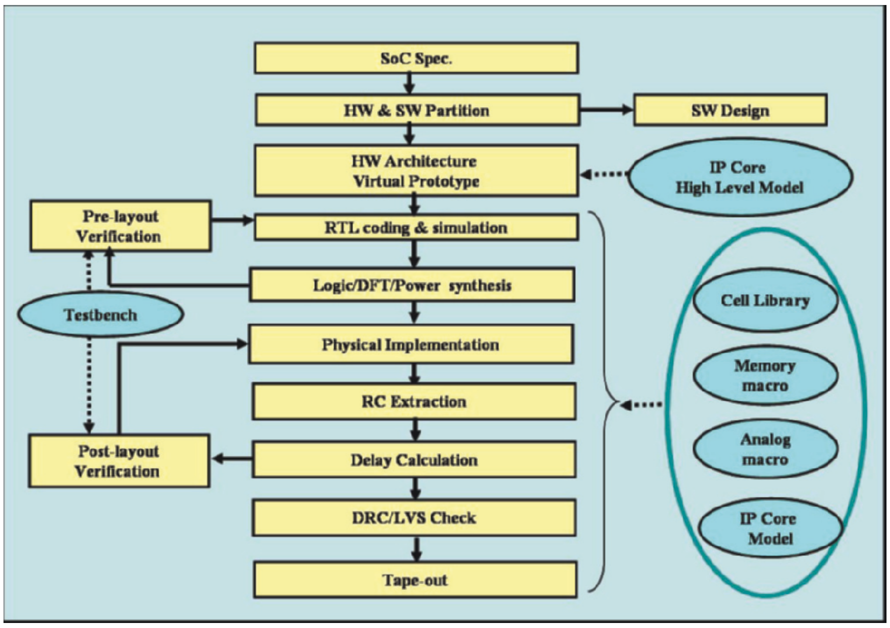


Figure 4. RTL to GDSII design flow

script was available and the first RTL level simulation was failed. We had to co-work with the IP vendor over 10 versions of RTL code modification or synthesis constraint updates.

- Because there is no automation tool available, we manually performed many versions of pin assignments to reduce the number of substrate layers from four to two resulting in packaging cost saving.

After overcoming all of the above problems, we were able to tape-out on time. Figure 5 shows the layout image of the chip. We achieved the first silicon work.

During mass production, manufacturing tests uncovered that the yield killer (5% loss) was in the insufficient driving strength of an output buffer in the CPU. The chip also went through reliability tests including ESD performance test, temperature cycle test, high/low temperature storage test and humidity/temperature test.

The mass production yield was enhanced from 82.7% initially to very close to the foundry yield model of 93.4% over a period of 8 months. Our measures include optimizing probe card overdrive spec, optimizing power relay waiting time, and retargeting Isat and Vth by optimizing poly CD in the foundry according to results from corner lot splitting.

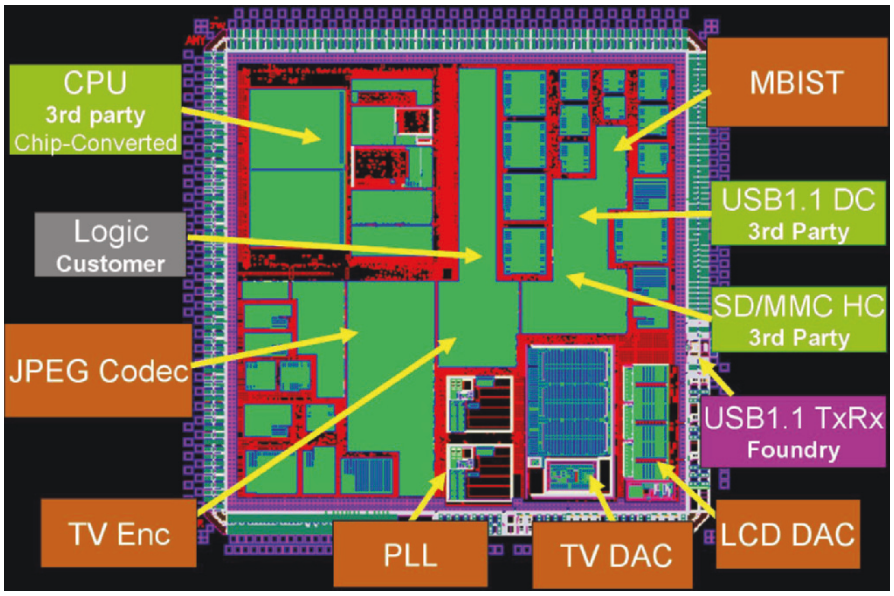


Figure 5. Layout drawing of the DSC controller SOC

We have been asked to perform failure analysis on 20 returned chips that have pins shorted to GND. After checking substrate delaminating and popped-corner using scanning acoustics tomography, we found no abnormality. Finally, by sinking 400mA of current to the corresponding pin of a good chip we concluded that the failure was due to a system board bug.

4. RECENT DEVELOPMENT

We went on to produce over three million of the chips over 18 months. Our system customer was able take about 8% of world-wide market share in the 2 and 3 million pixels segment during that period. We have also migrated the chip from 0.25um process to 0.18um one, achieving 20% saving in die cost. The migration was easy because we have been familiar with the design and the design flow.

The project has demonstrated that it is feasible to bridge the gap between the need of an electronics system house without IC design capability and the production capacity of a semiconductor foundry with an SOC design service provider. We have been able to leverage the experience gained and lesson learned to serve more customers and more projects such as DVD player, cellular phone set, electronics photo display, etc.

As both applications and technology become more advanced, we have expanded our IP portfolio to include MPEG-4 Encoder/Decoder/Codec, USB2.0 Device Controller, USB On-The-Go (OTG), SerDes I/O and embedded non-volatile memory such as flash and one-time-programmable (OTP). We have also enhanced our EDA flow to be able to simultaneously handle dozens of multi-million gate design at 0.13um and 90nm processes.

Current complex SOC projects require virtual prototyping, signal integrity check (crosstalk, electron-migration, dynamic IR drop, de-coupling cell insertion), design for manufacturability (intra-die process variation modeling, double via, dummy metal insertion), STA sign-off with in-die variation analysis, hierarchical DFT and physical implementation, low power solution (multi Vt/VDD cell library, gated clock, power down isolation) and flip-chip solution.

We have extended the development of memory BIST to more complicated SOC testing. In an SOC employing multiple IPs, each with test sequence, effective integration of all tests with necessary additional circuitry and test schedule is very important.

We have also extended the multimedia IP development from JPEG to JPEG2000, MPEG-4 and H.264/AVC standards. Although there are many software or ASIP approaches, we focused on pure hardwired approach because cost is a very important factor in mass consumer market. Figure 6 depicts the block diagram of an H.264/AVC decoder.

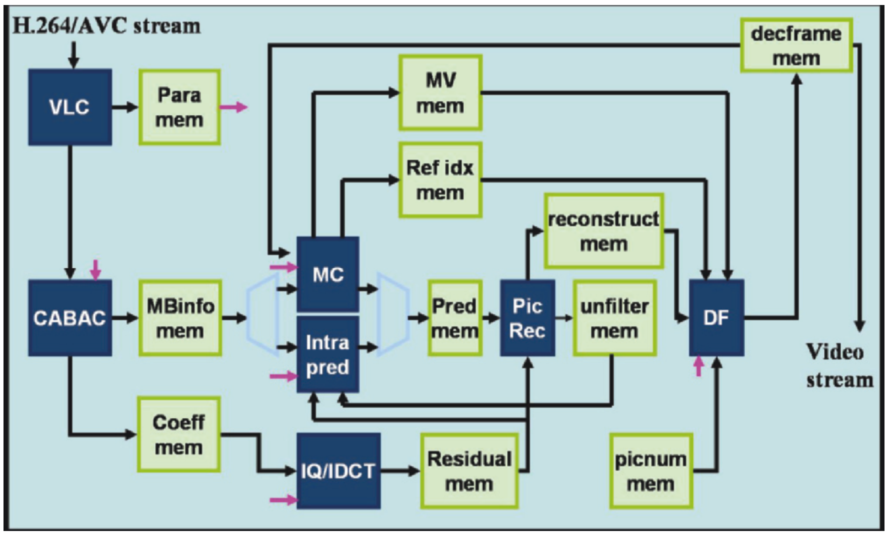


Figure 6. An H.264/AVC main profile video decoder

5. CONCLUSION

We have presented a new business model called SOC design foundry along with a case study of putting together resources and IP from both industry and academia, from multiple countries to implement a successful SOC for digital still camera all the way to mass production.

As applications are becoming more demanding and process technology is becoming more advanced, we expect to see more and more complex SOC integration. We will see advanced video such as H.264/AVC and wireless communication function being integrated together. Dealing with more IP sources is certainly more complicated but unavoidable.

A mass-production-proven SOC platform including IP, system, chip implementation to GDSII, and production methodologies will be a feasible approach to response to the challenge.

REFERENCES

1. C. L. Chen, J. Y. Lin, and Y. L. Lin, "Integration, Verification and Layout of a Complex Multimedia SOC," DATE-2005, Munich, Germany, March 2005.
2. C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 Still Image Coding System – An Overview," IEEE Transactions on Consumer Electronics, 2000.
3. C. J. Lian, Y. W. Huang, H. C. Fang, Y. C. Chang and L. G. Chen, "**JPEG, MPEG-4, and H.264 Codec IP Development**" DATE-2005, Munich, Germany, March 2005.
4. C. W. Wu, "SOC Testing Methodology and Practice," DATE-2005, Munich, Germany, March 2005.
5. G. K. Wallace and M. Maynard, "The JPEG Still Picture Compression Standard," Communications of the ACM, 1991.

Chapter 3

MULTIMEDIA IP DEVELOPMENT

Image and video codecs

Liang-Gee Chen, Chung-Jr Lian, Ching-Yeh Chen, and Tung-Chien Chen
Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, 1, Sec. 4, Roosevelt Road, Taipei 10617, Taiwan, ROC

Abstract: Multimedia intellectual property (IP) cores play a critical role in a successful multimedia SOC design. This chapter will focus on the design of image and video codec IPs, which usually requires lots of computational power. From theory to practice and from algorithm to hardware architecture, design methodologies toward an optimized architecture and also real design cases will be presented. Both top-down system analysis and bottom-up core module design are emphasized. Following theoretical discussions of the overall scenario, key building blocks of image and video codecs proposed in literature are reviewed. Examples will cover motion estimation, discrete cosine transform, discrete wavelet transform, and entropy coder. Then, complete image and video codec designs are explored. JPEG, JPEG 2000, and H.264/AVC are the three case studies. This chapter is intended to provide an overview, from theory to practice, on how to design efficient multimedia IPs

Keywords: image; video; compression, codec; architecture; intellectual property (IP)

1. INTRODUCTION

In this chapter, the design issues and methodologies of image and video codec IPs are discussed. Driven by cost and performance, system-on-a-chip (SoC) is a design trend. Intellectual Property (IP) integration is a must for designers to bridge the gap between design productivity and technology advances. In the post-PC era, there are more and more multimedia consumer products. In a complex multimedia SoC, the development of an optimized image/video codec IP is a critical point.

Digital image/video compression and decompression require many computing and bandwidth resources. To cope with the design challenges of high-specification image and video codecs, dedicated architecture is chosen to provide the most efficient implementation. No matter the final integration is in the form of a platform-based design with dedicated accelerators in module level or a fully hardwired codec system, dedicated hardware does efficiently off-load the processor in a complex SoC.

This chapter is organized as follows. Section 2 is a brief introduction of digital image and video coding. Section 3 is a comprehensive discussion about the design issues and methodology for the development of a good codec IP. In Section 4, some module-level design cases are presented. These critical modules are the basic building blocks of a codec system. In Section 5, the JPEG [1], JPEG 2000 [2] and H.264/AVC [3] codec designs are discussed. Finally, Section 6 summarizes this chapter.

2. DIGITAL IMAGE AND VIDEO CODING

2.1 Applications

We start by talking about applications since it is the application that drives the advances of technologies. There are more and more multimedia products in our daily life. Consumers continue to look for not only convenient but also fancier appliances (Figure 1), such as digital still camera (DSC), digital camcorder, multimedia phone, DVD player, digital TV, etc. Digital image and video become one of the most attractive features.

As we continuously pursue higher quality digital image and video, the huge amount of digital image and video data become a problem. Unlike

Figure 1. Digital image and video applications

voice or text data, whose data size is not that large, both the transmission and storage of image and video data are big issues since high resolution and high quality image and video always result in large data size. To transmit or store the uncompressed raw data is wasteful in the view of time and cost. To alleviate these problems, image and video compression are important enabling technologies for multimedia products.

Thanks to the advances of IC technologies, modern multimedia products can be light, thin, and small. In the old days, a system consists of many chips. Nowadays, more functions can be integrated on a single chip. The form factor of IC and hence the overall product become smaller. Less cost and higher performance are achieved. Successful and on-going examples such as

- DVD chips that integrate the MPEG core, the servo control, and related signal processing.
- DSC chips that integrate the JPEG, the image processing pipeline, and the camera control.
- Multimedia phone chips that integrate the multimedia engine (audio, video and graphics), the base-band processor and system control.

2.2 Image and Video Coding Basics

The basic concept of image and video compression is redundancy removal. The types of redundancy can be classified as spatial, temporal, statistical, and visual redundancy. By some mathematical algorithm and human visual system (HVS) characteristics, the digital image and video information can be manipulated and represented in a more compact way. That is what digital image and video coding (compression) does to shrink the data size.

It will be a long story to talk all the techniques on image and video compression. Readers are referred to some other books [4][5][6] that have more detailed introduction of image and video coding algorithms and standards. This section only provides a brief overview of some basic techniques that are more representative. In the following, transform coding, quantization, entropy coding, motion estimation (ME) and motion compensation (MC) will be briefly introduced.

2.2.1 Transform coding: Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT)

Transform coding is to transform the image data from the spatial domain to the frequency domain. After the transformation, there is an advantage of signal energy compaction, which is better for data compression. Also, the HVS characteristic of the sensitivity on different frequency components can be used for quantization.

Transform coding forms the basis of image and video coding standards. For image coding standards, JPEG selects DCT [7] as its transformation, and JPEG 2000 adopts DWT. For current video coding standards, DCT is the mainstream. The DWT is adopted for the temporal filtering in the emerging scalable video coding (SVC) standard.

For an 8 × 8 block *x(m,n)*, where 0 ≦ *m, n* < 8, the forward and inverse 2D DCT equations are

$$Z(p,q)=\frac{1}{4}\quad(p)\quad(q)\sum_{m=0}^{7}\sum_{n=0}^{7}x(m,n)\times\cos\frac{(2m-1)\pi p}{16}\times\cos\frac{(2n-1)\pi q}{16} \qquad (1)$$

$$x(m,n)=\frac{1}{4}\sum_{p=0}^{7}\sum_{q=0}^{7}\quad(p)\quad(q)Z(p,q)\times\cos\frac{(2m-1)\pi p}{16}\times\cos\frac{(2n-1)\pi q}{16} \qquad (2)$$

where 0 ≦ *m, n, p, q* < 8, $\alpha(0) = 1/\sqrt{2}$, and $\alpha(i) = 1$ for $i \neq 0$. Figure 2 (a) shows the result of a Lena image after 8 × 8 DCT. The *Z(0,0)* of each block is the DC coefficient, whose energy is usually higher. The other 63 coefficients are AC coefficients, which contain higher frequency information. Their values are usually small, and may be quantized to zero at high compression level.

The 2-D DWT is a series of low pass, high pass filtering and subsampling in both horizontal and vertical directions. The spatial domain data are transformed into the LL (horizontal low pass, vertical low pass), HL (horizontal high pass, vertical low pass), LH (horizontal low pass, vertical high pass) and HH (horizontal high pass, vertical high pass) sub-band signals

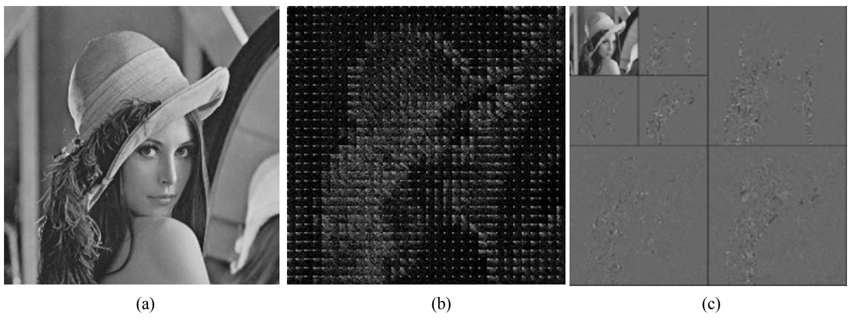


Figure 2. Transform coding. (a) original Lena image (b) Lena image after 8 × 8 DCT (c) Lena image after two-level DWT

in the frequency domain. In image coding, Mallat structure is usually adopted. That is, the LL sub-bands in each resolution can be further decomposed into four sub-bands. Figure 3 shows a 2-level 2-D DWT dataflow, and the result of a Lena image after the DWT of two-level dyadic decomposition is shown in Figure 2 (b). In JPEG 2000 Part 1, two filters are supported. The (5,3) filter is for lossless coding and the (9,7) filter is for lossy coding.

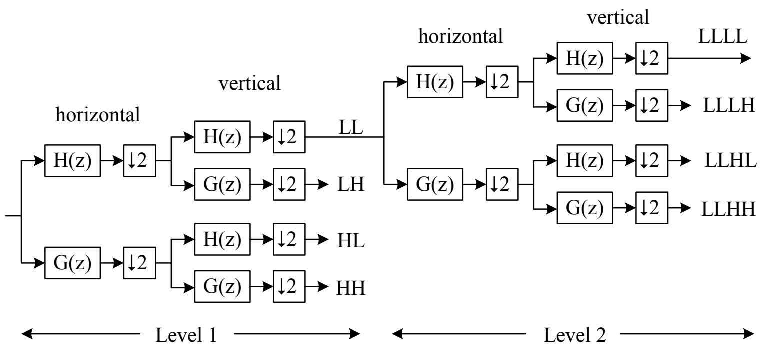


Figure 3. Two-level 2-D DWT. H(z): low-pass filter, G(z): high-pass filter

2.2.2 Quantization

Quantization is the main scheme to control the compression ratio. Lossless compression can only achieve limited compression ratio. By quantization, the range of compression ratio is widened, and can be adjusted by specifying different quantization extent.

In JPEG, 8×8 quantization matrices are used, and each entry of the quantization matrices can be specified by a user. The uniform quantizer of JPEG is defined as *Zq(m,n) = round(Z(m,n) / Q(m,n))*, where Z(m,n) is the DCT coefficients, Q(m,n) is the quantizer step size, and Zq(m,n) is the quantized DCT coefficient, normalized by the quantizer step size. The dequantization is defined by *Zdeq = Zq(m,n)* × *Q(m,n)*. For JPEG 2000, a specific quantization step can be defined for each subband. In MPEG video coding, the quantization step size is chosen by the quantization parameter QP defined in standards. In H.264/AVC, 52 different QPs are supported, and when the QP increases by one, the required data rate will decrease approximately 12.5%.

Quantization is a lossy operation where some information is selectively discarded and cannot be recovered at the decoding side. Therefore, there will be differences between the reconstructed image and the original one. The peak signal-to-noise ratio (PSNR) is a common index for objective quality evaluation. Quantization is based on rate-distortion model and HVS characteristics. Since human eyes are less sensitive to high frequency components, the quantization extent of higher frequency parts can be larger. In this case, the lost information is less apparent to human eyes.

2.2.3 Entropy coding: Huffman coding and arithmetic coding

Statistical redundancy can be removed by entropy coding. It is a lossless coding process based on the concept that more frequent symbols can be assigned shorter code words, and less frequent ones can be assigned longer code words. The average code length of the variable length coded data will therefore be shorter than fixed length codes. Huffman coding and arithmetic coding are the two main entropy coders used in image and video coding standards.

The implementation complexity of a Huffman coder is less than that of an arithmetic coder, while the compression performance of an arithmetic coder is usually better than a Huffman coder. In baseline JPEG [8] and MPEG-1/-2/-4, the Huffman coding is adopted. In JPEG, user-customized Huffman tables are supported, while in video coding, Huffman tables are fixed and predefined in the standards. In JPEG 2000 [9] and MPEG-4 Visual Texture Coding (VTC) tool, the binary arithmetic coding is adopted. The latest H.264/AVC standard supports both Huffman coding and Arithmetic coding as its coding tools. In baseline profile, context-based adaptive variable length coding (CAVLC) is supported, while in main profile, context-based adaptive binary arithmetic coding (CABAC) is adopted.

2.2.4 Motion Estimation (ME) and Motion Compensation (MC)

ME and MC are the most important techniques for the inter frame video coding to remove the temporal redundancy. They provide tens to hundreds more compression ratio compared with intra-only techniques. In a video sequence, the successive frames are similar since the time period between them is short. For a 30 frames per second video, the time differences between two frames are 1/30 second. The concept of ME and MC is to find a predictor in the reference frame(s) that can best predict the current frame data, and therefore, compensate the frame differences.

Block-matching ME is adopted in all video coding standards to find the best matched prediction data. A current frame is divided into macroblocks (MBs), and each MB in the current frame (current MB) is matched within the search range of the reference frame (Figure 4) by a matching criterion. The

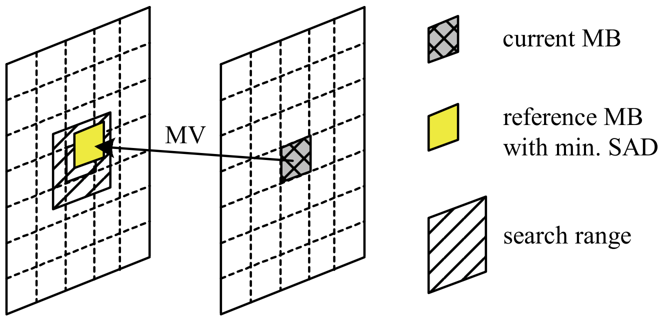


Figure 4. Block-matching motion estimation to find the motion vector (MV) of a MB.

sum of absolute differences (SAD) between a current MB and a reference MB is usually adopted as the matching criterion, which is defined by

$$SAD(k,p) = \sum_{i=1}^{N}\sum_{j=1}^{N} \left| cur(i,j) - ref(i+k, j+p) \right| ,$$

where N is the block size of a MB, *cur(i, j)* is the pixel value in the current MB, *ref(i+k, j+p)* is the pixel value in the reference block, the search range is $[-P_H, P_H)$ and $[-P_V, P_V)$ in the horizontal and vertical direction, and (k, p) is the position of the search candidate (a reference block) in the search range, $-P_H \leqq k < P_H$ and $-P_V \leqq p < P_V$. After the search, the search candidate with the smallest SAD is selected as the best reference MB, and the associated MB position is the motion vector of this current MB. The motion vectors are variable length coded, and the prediction error (residue) between the current MB and the reference MB is coded by JPEG-like intra coding.

2.3 Standards

The standardization of image and video coding algorithms make data exchange easier. We have briefly described some basic compression techniques in the previous subsection, and there are actually more techniques than those basic techniques. A proprietary algorithm can be a combination of any basic component among them. Interoperability becomes an issue when we want to share the compressed data with others.

Interoperability of the coded data is a key issue of the product popularity and cost. Therefore, international organizations start to standardize image and video coding standards. In the digital image field, JPEG should be the best model of standardization. It is so successful and popular that current DSCs all support JPEG compression. As for the digital video field, the big success of MPEG-2 is another good model. The DVD market is growing rapidly for the big entertainment requirement.

The advances of digital image and video coding standards keep going. The JPEG, MPEG and VCEG under ISO/IEC and ITU-T international organizations are the three groups consisting of many image and video experts who have long been devoted to the development of coding algorithms and standardizing them. Figure 5 shows the progress of some classical and state-of-the-art standards. Different standards focus on different applications. The trend is that the compression performance advances at the cost of higher complexity. Also, more features and functions are provided to fulfill the demands.

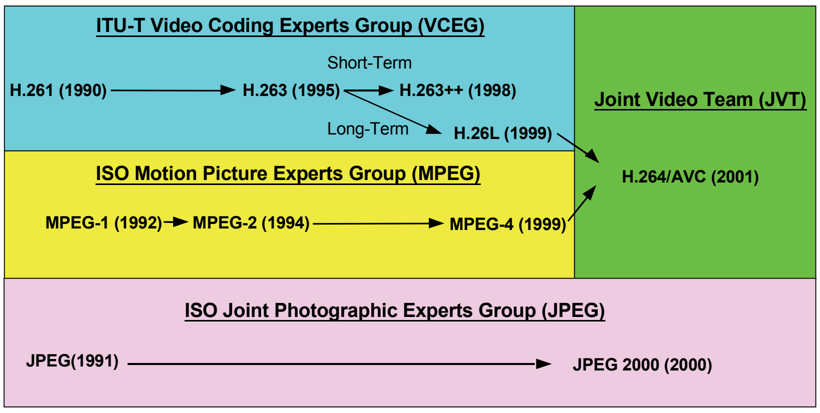


Figure 5. Advances of image and video coding standards.

Figure 6 shows the basic framework of image coding. Baseline JPEG [8] is a DCT and Huffman coding based coder, while JPEG 2000 [9] and MPEG-4 VTC is based on DWT and arithmetic coding. Figure 7 is the basic framework of a video coder. Besides the intra frame coding part (DCT, quantization and entropy coding), which is similar to a still image coder, the ME/MC is used for inter frame coding, and all this forms the hybrid coding architecture of most MPEG and H.26x standards. There is a decoder embedded in a video encoder, and a coding loop is formed in a video encoder. This is just a basic and simplified diagram. Different video coding standards have some specific features on some functional modules. Besides, in standards like H.264/AVC, there is an in-loop de-blocking filter.

The standard does not standardize everything. The scope of these image and video coding standards is only the detailed definition of the syntax and semantics of the bitstream and the decoding process. Standards still leave large room for the optimization of a codec design.

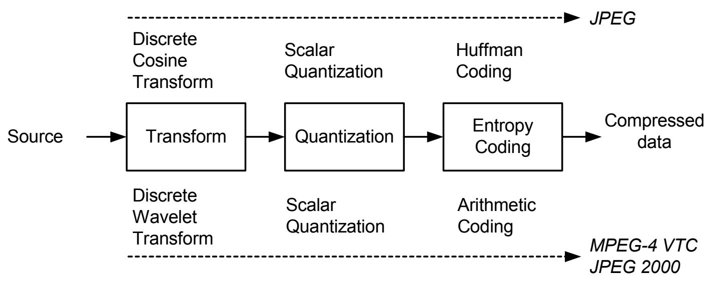


Figure 6. Basic framework of image coding

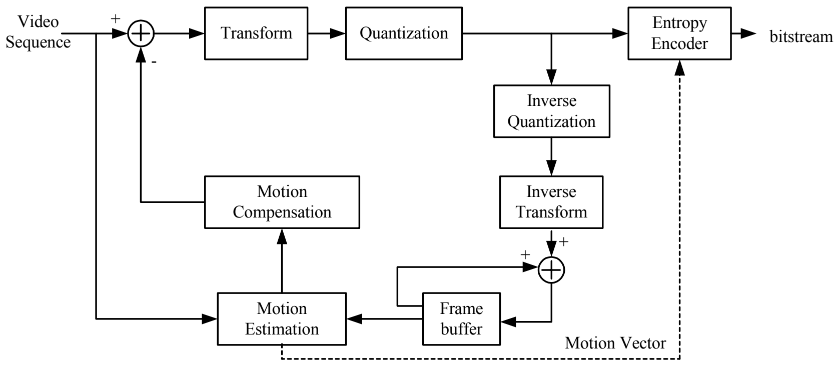


Figure 7. Basic framework of MPEG video coding standards

2.4 Characteristics of Image and Video Coding

The first step to design a good image/video codec is to understand the characteristics of image and video coding.

The standardized coding flows do not mean the standardized codec implementation. The design of a good image/video codec is not just a trivial mapping of standard algorithms to architectures but an optimization problem of timing, cost, power, etc., that requires many efforts. Besides the general IC design knowledge and techniques, we need to have an insight of the characteristics of the video data we need to process and the algorithms we are going to use. With deeper domain knowledge of image and video coding, designers can design better codec architectures.

Since an image or video codec has to process large raw data and compress them into smaller size, a codec itself also faces the large storage and bandwidth problem. For image encoding, the input is a huge amount of raw data. For an $M \times N$ 24-bit color image, there will be $M \times N \times 24$ bits or $M \times N \times 3$ bytes. If the image is sub-sampled to YUV 4:2:0 format, the size becomes $M \times N \times 1.5$ bytes. Take a 5-million pixels DSC for example, one raw image size is about 7.5 Mbytes. Unlike video's real time requirement, there is no exact time budget for the processing of an image. The guideline is to process an image as fast as possible to avoid the compression engine becoming the bottleneck, compared with other system components such as flash memory access time.

For 30 frames per second (4:2:0, resolution M × N) video data, the input data rate will be $M \times N \times 12 \times 30$ bits/s. For real time applications such as video conferencing and broadcasting, the compression and decompression of a frame has to be done within 1/30 second. Also, the processing delay should be kept as low as possible.

Different modules have different operation characteristics. In the transform and motion estimation stage, it is a block-based operation. Image and video data are partitioned into blocks. Typical DCT block is of size 8 × 8, and ME macroblock is of size 16 × 16. The computational complexity of transform coding and motion estimation dominates the video codec. These operations are more regular since operations basically are done in the block-based coding units. Therefore, parallelism exists inherently in these algorithms, and high parallel array processor can handle this loading. In the later entropy coding stage, usually it is a bit-level processing. There are no complex mathematical operations but fine and delicate variable-rate and variable-length data processing.

Besides the objective analysis of data and algorithm characteristics, human visual perception also plays an important role on image and video coding. The key is that the video is for human eyes, not for computers. What people see is a beautiful image, not the binary digits and numbers. Since human visual system (HVS) is more sensitive to low-frequency signals than high-frequency ones, quantization is based on this characteristic. Also, the trade-off between computational complexity and quality is feasible.

In summary, for the image/video codec design, we see something good:

- Standardization of algorithms
- Regular and simple computation: DCT/DWT, ME, … except entropy coding
- Regular data flow
- Human perceptual tolerance

At the same time, we see something bad:

- Real time requirements especially for multimedia communications: scheduling, timing
- Resource limitation, especially for portable applications: computing power, battery energy, storage, channel, …

- High data rate
- Multi-mode multimedia requirements: JPEG, JPEG 2000, MPEG-1/-2/-4, H.263, H.264/AVC, SVC, MP3, AAC, CELP, …

3. DESIGN METHODOLOGY

3.1 System Analysis

Before hardware architecture design, system analysis was the most important step to have an insight into the design problem designers will face. The goal of system analysis is to find out the bottleneck of a video codec design so that designers can focus on it and get an optimal design.

Computational complexity and memory access are the basic and important data we want to get from system analysis. There are many different approaches and tools for system analysis. For a designer who is somewhat familiar with the video coding concept and algorithm, he can have rough but good enough estimation by hand calculation analysis. A more general approach is to use a general processor (PC or workstation) for software profiling. Although the analysis results differ when the software is running on different platforms, the analysis data still provide designers a good starting point to understand the complexity of the system and each module. The profiling tools can be, for example, common run time profile tools on workstation and PC, instruction profiling tools, iprof [10], or Intel® Vtune^{TM} Analyzer, etc. It is important to know what we want through the analysis data. If the final implementation is not on these general processors, the profiling data from that are just for reference. The percentage and the order of those numbers are more important than exact numbers.

3.2 Architecture Exploration

3.2.1 Design alternatives overview

The design space of signal processing ICs is wide and colorful. There have been various hardware architectures explored in the literature, and the exploration steps still keep going. Although some variants may exist in different architectures, the design alternatives can be mainly classified as follows. The two extremes of the design space are processor-based design and fully application specific IC (ASIC) design. In between, there are architectures such as platform-based, FPGA, digital signal processor, multimedia processor, application specific DSP, and some others that combine different architecture as part of the system [11].

Processor-based design provides better programmability and lower performance for video coding, while the ASIC design provides best performance with little flexibility. General purposed processors are too general to handle the loading of video coding. Adding application specific instructions and/or dedicated accelerators can greatly improve its performance for video coding. Approaches like that are application specific DSP (ASDSP) and video/media processors. At the same time, general FPGA also try to embed more dedicated arithmetic units to boost their capability.

3.2.2 Application specific architecture

There are many arguments over the pros and cons of different architectures. Basically, what an optimal architecture should be depends on applications and the overall system considerations. The guideline is to provide just enough computing resources at the lowest design cost and time. In this chapter, there is not enough space and it is not our intention to introduce the whole design spectrum, either. What we will focus on are the analysis and design on the dedicated architecture for the key modules and possible complete codecs.

An image/video-specific architecture is optimal for computational performance. Digital image and video, as killer applications, deserve a special treatment. Video coding needs to process many data, and the complexity of algorithm keeps soaring. Even for a processor- or platform-based design, designers also have to understand the design of a dedicated architecture so that they will be able to enhance their processor architecture. Therefore, in the following part of this chapter, the dedicated architecture design of a video codec is discussed. Though the content is more ASIC oriented discussions, the ideas in dedicated architecture design actually are also foundations for other programmable design alternatives to enhance the performance with specific instructions, co-processors, or accelerators.

3.3 Design Issues and Techniques

3.3.1 Speed

For image and video applications, the speed requirement is usually the most urgent issue. Actually, for on-line video applications, the first priority of an implementation is to meet the real-time specification.

Since there is a large amount of data to be processed within a tight time constraint, general processors, which execute the computations sequentially, cannot afford such high computational load. Only raising the working frequency is not a good approach to solve the problem. Working fast but not efficiently is of course not an optimal way. Also, high power consumption is a problem at higher frequency especially for battery-powered portable appliances. Therefore, for image and video codec designs, ASIC designers usually do not pursue very high operating frequencies. On the contrary, designers look for more efficient architectures to be operated at lower clock rate. SIMD (Single Instruction, Multiple Data), VLIW (Very-Long Instruction Word), and array processor designs are examples of higher computational efficiency.

3.3.2 Area

IC designers always look for compact architectures since smaller die area means lower cost. A more compact design will have better competitiveness. In a dedicated accelerator design, parallel architecture is usually adopted to achieve the required specification while lowering the required working frequency. Area is used to trade with frequency.

Parallel architecture results in higher area cost. Therefore, hardware utilization is an index that designers should take care of and check. Simply duplicating multiple processing elements and memories may not be an optimal way if some of these resources are with low hardware utilization rate. The optimization goal should be a just enough parallelism with as higher as possible hardware utilization. Algorithm-level optimization, hardware sharing and folding are example techniques for chip area optimization.

The fact that process technology keeps improving rapidly is really good news for parallel architecture design. Although area cost is always an issue, the weight per transistor becomes lower and lower in a million or even billion transistors level SoC. Actually, designers now are facing a bigger problem on how to bridge the gap between the design and the process capability, not just keeping an eye on the minor optimization of several hundreds or thousands gates.

3.3.3 Power

The power issue becomes one of the most important problems in a SoC design. As with the process development, a chip can provide more and more transistors, but the allowed power consumption in a chip will not increase. The emerging portable multimedia devices ask for more restricted power consumption. Besides, the heat due to high power consumption will also cause the reliability problem.

To cope with the power problem, a design should be carefully examined not only in architecture level but also in algorithm and circuit levels.

Actually, the higher-level optimization usually provides more gain in power saving. The development of fast algorithms with lower complexity, and algorithms with lower data bandwidth are examples of this. Designers have to understand the algorithm characteristics by detailed analysis.

At architecture level, designers have to look into the detailed operations of each module or even each gate, and its power consumption behavior. High hardware utilization architecture will be more power-efficient since the power will not be wasted on idle gates. Memory is also a big source of power consumption. Therefore, memory hierarchy and arrangement will play an important role on a low power design. At last, the algorithm and architecture characteristics can be combined with circuit level techniques such as clock gating and dynamic voltage scaling, etc.

Besides the low power issue, power aware design is another trend. In image and video coding, there are multiple modes and tools. Also, different algorithms with different computational complexity results in different quality level. The basic power aware design concept is the rate-distortion-complexity optimization. That is, a codec can dynamically decide the operating points based on the available power budget. Take ME for example, different ME algorithms can be mapped to a reconfigurable architecture. This architecture then have multiple operating points that have different power consumption and quality of MV search.

3.3.4 Bandwidth and storage

Memory bandwidth and on-chip memory capacity are limiting factors for many multimedia applications. Today, in many designs, on-chip memory has already occupied more than 50% of total chip area. Good memory management and area-, power-, and yield-efficient memory implementations, become important for a successful SoC solution.

The memory management is to provide an efficient memory hierarchy that consists of off-chip memory, on-chip memory, and registers [12], as shown in Figure 8. Different memory types have different features. Off-chip memory, usually DRAM, offers a large amount of storage size but consumes the most power. The off-chip memory and I/O access may dominate the power budget. The embedded DRAM is developed to reduce the I/O access by integrating large on-chip DRAM. However, the embedded DRAM technology is not very mature because the yield issue, design methodology, and many physical design challenges still need to be solved. Besides, embedded compression (EC) technique can be applied to reduce the off-chip memory bandwidth and size for a video codec design. The on-chip memory, usually implemented by SRAM, provides faster access and less power-consumption than the off-chip memory, but the memory cell size is much

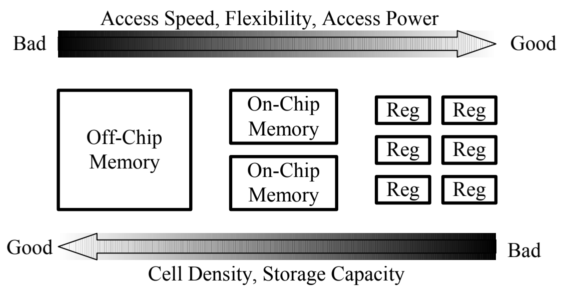


Figure 8. Memory hierarchy: trade-offs and characteristics

larger. Registers can be faster than on-chip memory and provide more flexible data storage. However, the size of a register is the largest. Registers are more suitable for the implementation of smaller-size buffers.

Memory management can be organized from two different levels: algorithm-level and architecture-level. The embedded DRAM is one kind of circuit-level improvement. But even integrating large embedded DRAM, the access power is still larger than smaller on-chip SRAM. The algorithm-level memory hierarchy optimization is to modify the coding system algorithm to improve some system parameters, like power or area, and some other parameters, like coding performance, become the trade-off. The EC and hierarchical search ME belong to algorithm-level memory optimization. On the other hand, the architecture-level memory organization can optimize the memory hierarchy from modifying hardware architecture, such as Level A to D data reuse schemes in ME architecture design that will be discussed in a later section in this chapter.

Video codecs face a big problem of the increasing bandwidth and storage requirements. The resolution of image and video keep increasing since people are endlessly pursuing higher quality. The input and output of image data itself has already been a large bandwidth request. Besides that, the module such as ME consumes even more bandwidth due to the block-matching process among multiple image frames. Since the data moving between modules and modules and between modules and memory is so frequent, dedicated bus between modules is a better solution and local memory is required to reduce the bandwidth requirement between modules and memory. Since the trend shows that memory may dominate most of the designs, it is therefore important to optimize the memory size not only because of the area issue but also for lower power consumption.

3.3.5 Perceptual quality

Video is for human eyes, not for computers. The perception of human eyes determines the quality requirement of video signal processing. For digital video, larger dynamic range of each word means better quality, but with larger hardware cost. Designers have to make decisions based on product specification. Another good reference is the saturation point of quality improvement when the bit width is increasing.

Besides the trade-off between area cost and quality, computational complexity can also be lowered at some cost of quality degradation. ME is the most time consuming part. Instead of the full search, there are many possible lossy ways for ME. These fast ME algorithms can dramatically reduce the required computational power, while the quality is still kept within an acceptable range.

4. MODULE-LEVEL DESIGN

An image/video codec system can be intuitively partitioned to modules based on the functionality. Although there are different image and video coding standards, the basic framework is similar. For image coding, standards like JPEG, JPEG 2000, and MPEG-4 Visual Texture Coding (VTC) are all composed of transform, quantization, and entropy coding modules. As for video coding standards, such as H.261, H.263, MPEG-2, MPEG-4, and H.264/AVC, they are motion compensation (MC) and DCT based hybrid-coding framework.

Since most image and video codecs share the same or similar key components like DCT/DWT, ME, and entropy coder, modulized design concept is usually highlighted for module-level IP optimization and design reuse. For a fully dedicated codec design, each module is mapped to hardwired architecture, and the system is built based on these functional modules to form a processing pipeline. For a platform-based design, key modules can be accelerators attached to the system bus and controlled by a processor.

4.1 Design Issues

The most critical module deserves more efforts on optimization. Also, different modules have different operation characteristics, so different techniques should be applied for optimization. Although we are discussing the module level design here, designers should always keep the system view in mind. The integration of several so-called optimized modules at module level may not always guarantee an optimized architecture at system level.

To start designing a module, designers have to have a specification from a system. Then, a detailed analysis about the complexity and operation type is the key before architecture design.

Parallel and pipeline are the two general and basic but important techniques for architecture design. The specification and complexity will determine how many parallelisms are needed. The computational flow is then smoothed and pipelined to have better timing performance.

Interface among modules is important since each module has to work with other parts of the system. Sometimes, the interface considerations will be more critical than the internal design of a module. If, for example, each module pushes some of the design problems to external world, each module is only optimized at that assumed environment. The integration of these modules may pay more costs on the interface design to connect them.

In the following sections, four main functional modules are discussed. The first topic is the ME. It is the most time-consuming part in a video encoder. The second topic is transform. Both DCT and DWT will be included. The third topic is about entropy coding and decoding modules, including Huffman coder and Arithmetic coder. Finally, the design of bitstream parser in a decoder is introduced.

4.2 Motion Estimation

Motion estimation is the core of a video codec.

The design of a motion estimator is so attractive because of two characteristics: high complexity and high flexibility. The operations of ME are regular but the computational complexity of ME is very high. For a software codec, the ME is the most time-consuming module, and for a hardware codec, the ME consumes most of the resources including gates, bandwidth and power. It is so critical that an optimized ME architecture usually will dominate the factors in a successful video codec architecture design.

ME is flexible in algorithm level, since video standards only specify the decoding part. How the motion vectors are searched, what the matching criterion is, and which candidate macroblock should be the chosen one are not standardized. Therefore, the large room for the development of fast algorithms and the application of proprietary tricks make the design space of a motion estimator even broader.

The ME design [13] deserves the first-priority concern and more space for discussions. We will describe several types of ME algorithms first. Then, the associated architecture designs are presented, followed by a discussion about the bandwidth issues in a motion estimator.

4.2.1 Algorithms

The block-matching ME [13] is composed of seven loops, as shown in Figure 9. The first loop, frame-level loop, is the number of frames in a video

```
for Number-of-Frame   ( Frame-level loop )

  for Number-of-MBv   ( MB-level loop in the vertical direction)
  for Number-of-MBh   ( MB-level loop in the horizontal direction)

      for Number-of-SRv   ( SR-level loop in the vertical direction)
      for Number-of-SRh   ( SR-level loop in the horizontal direction)

          for Number-of-CBv   ( CurBlock-level loop in the vertical direction)
          for Number-of-CBh   ( CurBlock-level loop in the horizontal direction)
              ... ...
                ...
          end of Number-of-CBh
          end of Number-of-CBv

      end of Number-of-SRh
      end of Number-of-SRv

  end of Number-of-MBh
  end of Number-of-MBv

end of Number-of-Frame
```

Figure 9. The loops of a block-matching ME procedure

sequence. The second and third loops (*MB-level loops* in the vertical and horizontal directions) are the number of current MBs in one frame. The search region level loops (*SR-level loops*) are the number of search candidates in a search region, and the last two loops (*CurBlock-level loops*) are the number of pixels in one current MB for the computation of SAD.

For the full search block-matching algorithm (FSBMA), all candidates in a search range are examined, and the candidate with the smallest distortion in the search range will be selected as the final motion vector. Exhausted search guarantees a globally minimum SAD in the search range. However, the computation complexity is very high. For example, a real-time ME for CIF, 30 frames per second (fps) video with the (-16, 16) search range requires 9.3 Giga-operation per second (GOPS). If the frame is D1-size and the search range is (-32, 32), the complexity is increased to 127 GOPS. Therefore, many fast algorithms, which apply specific strategies in different loops, are proposed to reduce the required computational complexity.

In the following, several fast algorithms are reviewed. We start by describing the fast full search algorithms, which do reduce some computations of the full search while the block matching result is the same as full search. That is why we call it fast full search. Sometimes, it is still hard to achieve the real-time computation with fast full search, especially for a large frame size or search range. Therefore, fast search algorithms that require much less computations at a cost of certain extent of quality drop are developed.

4.2.1.1 Fast full search

Is it possible to reduce some computations of FSBMA but without any quality drop? The answer is yes. The main idea is to detect and skip unnecessary computations earlier in the *CurBlock-level loop* of the ME procedure. The partial distortion elimination (PDE) [14] algorithm and the successive elimination algorithm (SEA) [15] are two typical examples.

The PDE algorithm is based on the observation that during the search if the accumulated absolute difference (partial SAD) of this search candidate has already been larger than the current minimum SAD, this candidate is guaranteed not the optimal one. Therefore, the accumulation of the partial SAD for this candidate position can be terminated immediately and we can move to the next search candidate. The concept of PDE is simple and effective.

In SEA, the absolute difference between the sum of pixels in the current block and the sum of pixels in a search candidate is used as a criterion to help early skip some candidates so that the calculations of the SADs of these candidate blocks can be completely saved. It is based on the inequality

$$\begin{aligned} SAD(k,p) &= \sum_{i=1}^{N}\sum_{j=1}^{N}\left|cur(i,j)-ref(i+k,j+p)\right| \\ &\geq \left|\sum_{i=1}^{N}\sum_{j=1}^{N}cur(i,j)-\sum_{i=1}^{M}\sum_{j=1}^{N}ref(i+k,j+p)\right| \equiv S' \end{aligned}$$

If *S'* is larger than the current minimum SAD, then the SAD of this candidate block is also larger than the current minimum SAD. Hence, the computation of the SAD for this search candidate can be skipped. On the contrary, if *S'* is smaller than the current minimum SAD, the SAD for this search candidate has to be computed. The sum of current pixels in the current block is only computed once and can be reused for all search candidates. The sum of reference pixels for different search candidates can be easily calculated by reusing the partial result. Because the computational complexity of this detecting procedure is relatively small, the overall computational complexity of SEA-based full search can be reduced.

For both PDE and SEA, a good initial search candidate can provide a better computation reduction ratio. If the current minimum SAD is closer than the final minimum and found early, the reduction ratio will be higher. The motion vector predictor or spiral scan technique is usually adopted as an enhancement of the PDE algorithm and the SEA.

4.2.1.2 Fast search by the simplification of matching criterion

Fast search by the simplification of matching criterion is a CurBlock-level loop simplification. Subsampling [16][17] is an approach that not all pixels in the current block are used to calculate the SADs for each search candidate. Another approach is pixel truncation [18], which means a reduction of the number of bits of each pixel during SAD computation.

4.2.1.3 Fast search by the reduction on search candidates

The second type of fast search is the reduction of search candidates. It is a SR-level reduction. These algorithms assume that the distortion monotonically decreases as the search candidate approaches to the optimal one. That is, even if we did not match all the search candidates, the optimal search candidate can be achieved by following the search candidate with the smaller distortion.

This category is the major part of fast search algorithms. Typical variants are center-based diamond search [19][20], advanced diamond zonal search [21][22], three-step search [23], two dimensional logarithmic search [24], one dimensional full search [25], new three step search [26], four step search [27], block-based gradient descent search [28], predictive line search [29], and so on. The Diamond search is illustrated below as an example.

Figure 10 shows the searching procedure of diamond search, and the candidate search pattern. In the searching procedure, the large diamond is applied until the center search candidate of the large diamond has the smallest distortion among nine candidates, and then the small diamond is used to refine the searching result.

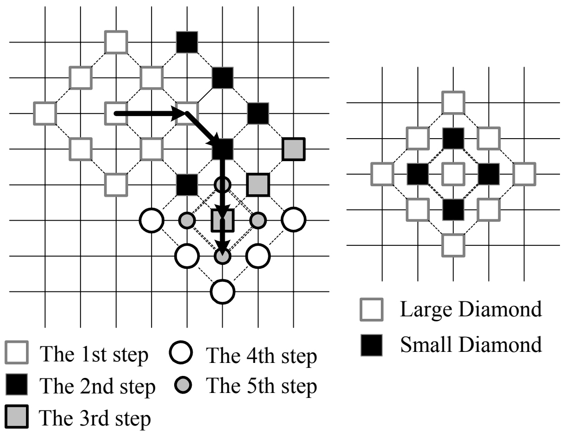


Figure 10. The search procedure (left) and the search pattern (right) of the diamond search. The arrow is the direction that a large diamond moves toward, and after the result of large diamond converges in the fourth step, the small diamond is adopted to refine the results in the fifth step

4.2.1.4 Predictive search

Predictive search helps fast search algorithms not to be trapped by local minimum SAD. It uses the motion information of neighboring blocks in the spatial or temporal space for the initial guess of the starting point. That is, the information of MB-level loop and Frame-level loop is utilized to predict the motion vector of this current MB and save the computations of SR-level loop. For example, the initial search candidate can be the motion vectors of the blocks on the top, left, and top-right, their median, zero motion vector, the motion vector of the collocated block in the previous frame, or the accelerated motion vector of the collocated block in the previous two frames. By this way, the search range can be reduced and constrained, so not only the computational complexity but also the bit-rate of motion vector can be saved.

4.2.1.5 Hierarchical search

Hierarchical search [30][31] is a multi-resolution search scheme. An initial estimation at the coarse level (subsampled resolution) is processed first, and then a refinement at the fine level is executed. Usually, two-level or three-level hierarchical search is adopted. At the coarse level, because of the subsampling in current MB and search region, the computational complexity becomes smaller and full search is usually adopted to find the optimal MV in the subsampled search region. Take the search result at the coarse level as the initial search candidate, the search range at the fine level can be reduced, and computational complexity can be saved. In general, hierarchical search is mostly adopted for high resolution and fast motion that requires large search ranges.

4.2.2 ME architecture

The computational complexity of ME is very high but the operations are quite regular. ME algorithm itself is with large extent of parallelism. Therefore, highly parallel array processor design is a common view for ME designers. An simplified functional view of ME architecture consists of two parts (Figure 11), the processing element (PE) array, which is responsible for the calculation of SAD, and the on-chip memory, which is used to store the data of search region and supports data reuse. For each MB, PE array will update the data in the on-chip Memory through data bus. After the required data are ready, PE array starts to compute the SADs.

Based on different ME algorithms and adopted search range data reuse schemes, the design of PE array, the size of on-chip memory and the memory bandwidth requirement will be very different. In the following, several typical ME architectures for different ME algorithms are introduced

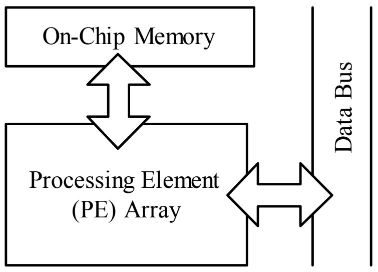


Figure 11. The simplified diagram of motion estimation architecture

first, where we assume for simplicity that the search range is (-2, 2) and the current block size is 2×2. After that, the trade-off between on-chip memory size and the required memory bandwidth in different search region data reuse schemes is discussed.

4.2.2.1 Full search ME architecture

Inter-level architectures and intra-level architectures are two types of full search ME architectures. The former is to compute the search candidates in parallel with the SR-level loop and sequentially estimate the distortions of all current pixels in the *CurBlock-level loop*. On the contrary, the latter is to compute the search candidates in sequential in the *SR-level loop* and parallel estimate the distortions of all current pixels in the *CurBlock-level loop*. In general, the former has a short critical path with a large register, and the latter has a long critical path but fewer registers. Moreover, the former requires fewer data inputs by broadcasting the reference pixels and propagating the current pixels, and the latter requires much more data input than the former does.

Figure 12 shows an inter-level architecture [32]. The inter-level ME architecture computes the search candidates in the SR-level loop in parallel, and sequentially estimate the distortions in the *CurBlock-level loop*. The PE in the inter-level architecture is responsible for the computing the differences of all current pixels in the current block and the accumulation of SAD for a candidate pixel by pixel. This PE array is a one-dimensional inter-level architecture that can compute all search candidates in a row at the same time. For example, in Figure 12 (b), there are four processing elements for four search candidates in a row, when the search region in the horizontal direction is (-2, 2).

The data flow is as follows. Current pixels are inputted in the raster scan order and propagated by the shift registers. Reference pixels are also inputted in the raster scan order to each reference pixel input, *Ref.Pixel0* and so on, and broadcasted into all PEs by the selection signals, *Sel0*, *Sel1*, and so on. In each cycle, each PE calculates the distortion between one current pixel and one reference pixel and accumulates this distortion to the partial

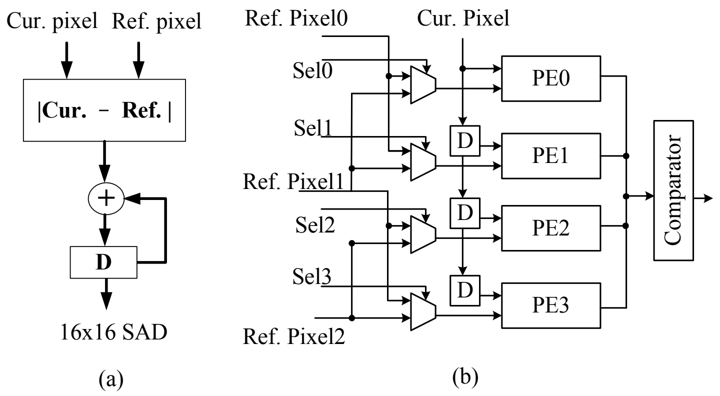


Figure 12. (a) The PE of inter-level ME architecture (b) the inter-level ME architecture

SAD of its search candidate. After 2×2 cycles, the first *PE*, *PE0*, will generate the SAD of the top-left search candidate. And in the following cycles, the SADs of search candidates from left to right in a row will be generated sequentially. The PEs which have generated the SADs in a row will process the search candidates in the next row until all search candidates are processed.

The intra-level architecture is another kind of architecture for FSBMA. In intra-level architectures, the current pixels in the *CurBlock-level loop* are processed at the same time, and the search candidates in the SR-level loop are computed one by one. The AB2 architecture in [33] is a two-dimensional intra-level architecture. The PE in this architecture is responsible for the distortion between one specific current pixel and the corresponding reference pixel for all search candidates, as shown in Figure 13 (a). Because AB2 is a two-dimensional intra-level architecture, there are four intra-level PEs, which are corresponding to 2 × 2 current pixels in the current block in this architecture, as shown in Figure 13 (b).

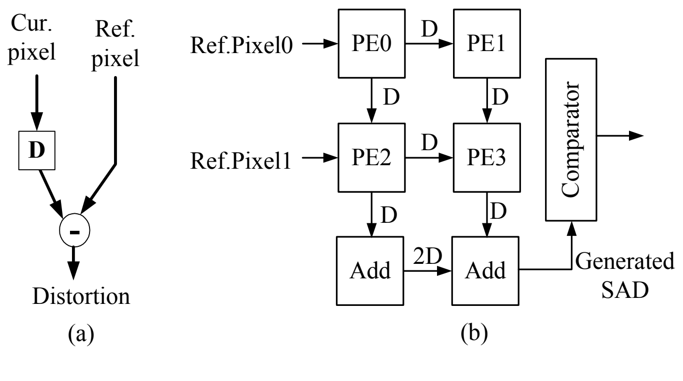


Figure 13. (a) The PE of intra-level architecture. (b) The intra-level ME architecture

The data flow of the intra-level architecture is as follows. Current pixels are stored in corresponding PEs, and reference pixels are propagated PE by PE in the horizontal direction. The two partial column SADs are propagated and accumulated in the vertical direction first. After the vertical propagation, two column SADs are propagated in the horizontal direction. In each PE, the distortion of a current pixel in current MB is computed and added with the partial column SAD, which is propagated in PEs from top to bottom in the vertical direction. In the horizontal propagation, two column SADs are accumulated one by one by two adders and four registers.

4.2.2.2 Fast search ME architecture

Although the computational complexity of fast search and fast full search algorithms is much smaller than that of FSBMA, the design challenges of VLSI architectures for fast search and fast full search algorithms is much more difficult than that of FSBMA. This is because the data flow of fast search and fast full search algorithms is irregular, and the processing order of search candidates is dynamic, which is dependent on the last searching result. For example, in three-step search or diamond search, you do not know the center position of the next searching step until the minimum of this searching step is found. Therefore, latency and pipelining bubble cycles are becoming critical issues. A good fast search ME architecture should have a short latency, support random access of search candidates efficiently, and have no pipelining bubbles cycles when skipping some search candidates.

Tree-based architecture [34] has the above-mentioned advantages. Figure 14 shows the tree-based architectures with different degrees of parallelism. The tree-based architecture is similar to the intra-level architecture. It can not only compute the distortions of the current pixels in the *CurBlock-level loop* in parallel but also process the search candidates in the *SR-level loop* at the same time, which means that the processing order in the *CurBlock-level loop* and *SR-level loop* can be reordered for different degrees of parallelism in tree-based architectures. For example, if the degree of parallelism in the tree-based architecture is two dimensions of current block, the tree-based architecture is equal to the intra-type architecture. If the degree of parallelism is only one dimension of current block, then it only processes the distortions of current pixels in a row at the same time. In the following, we take these two examples to illustrate the data flow of tree-based architectures.

In the first case, as shown in Figure 14 (a), each PE in tree-based architectures is corresponding to one current pixel and is responsible for the calculation of the distortion between one current pixel and one corresponding reference pixel for all search candidates. In this architecture, it can generate the SAD of one search candidate in one cycle. The latency of this architecture is dependent on the memory access of reference pixels,

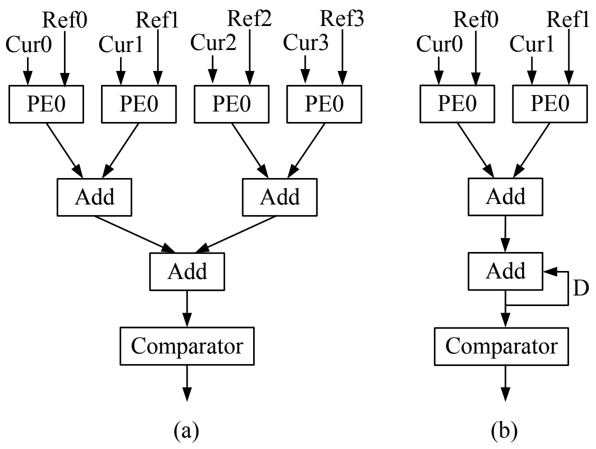


Figure 14. (a) The architecture of a tree (b) The architecture of a ½ tree

which can be shortened with the interleaved memory arrangement in [34]. Moreover, no pipelining bubble cycles and no data dependency between the current and the next search candidates exist in this architecture. It is suitable for the hardware implementation of fast search algorithms that require the property of random access in the search region.

If the degree of parallelism in the tree-based architecture is only one dimension or less of current block, as shown in Figure 14 (b), the tree-based architecture is a hybrid architecture of inter-type and intra-type architectures. For example, folding the architecture in Figure 14 (a) by 2 (N) derives the architecture in Figure 14 (b). Then, it can calculate one row SAD of the search candidates in one cycle, and after generating and accumulating all row SADs in 2 (N) cycles, the total SAD of one search candidate can be derived. In this architecture, PDE can be easily integrated. The comparison between the partial SAD and the current minimum SAD of PDE is changed from the distortion of one pixel to one row SAD.

The tree-based architecture has a good flexibility to support various reordering or rescheduling in the *CurBlock-level loop* and *SR-level loop*, so it is usually adopted for fast search algorithms. In [35], a tree-based architecture is adopted to support the diamond search and fast full search. There are many duplicated search candidates in diamond search, as shown in Figure 10. After each moving of the large diamond pattern, only five or three search candidates are required to be calculated and the others are calculated at the last large diamond pattern. An ROM-based solution is proposed in [35] to avoid the duplicated search candidates and save 24.4% search candidates in the Diamond Search algorithm.

This architecture also supports the fast full search algorithms, such as PDE and SEA. The computation of SEA for one search candidate is executed by the processor platform first and then the SAD of this search candidate is calculated in this architecture. The degree of parallelism in this architecture is only half row (*N*/2), so the PDE is easily integrated with less overhead, and the unit of comparison between the partial SAD and the current minimum SAD is changed to half row SAD.

4.2.3 Block-level data reuse for search region

The required memory bandwidth of ME is very huge. Several block-level data reuse schemes [36][37][38] for the search region have been explored and analyzed in the literature to save the required memory bandwidth. In the following, the redundancy access factor, *Ra*, is used to represent the required memory access of different data reuse schemes. The redundancy access factor, *Ra*, is defined as *Total memory bandwidth for reference frame / minumum memory bandwidth (pixel count in total)*, which means that if we want to process one current pixel, how many reference pixels are required.

The first scheme, Level A scheme, is the data reuse of the SR-level loop in the horizontal direction, and reuses the overlapped region between two reference blocks of two successive search candidates in the horizontal direction. As shown in Figure 15 (a), two reference blocks have a large common region, *N* × *(N-1)*, and only *N* pixels are different. Therefore, only *N* reference pixels are required to be updated for the next search candidate in the horizontal direction. Therefore, for a current MB, the required memory access, $Ra_{Level\,A}$, will be

$$Ra_{LevelA} \approx \frac{N \times (SR_H + N - 1) \times (SR_V)}{N \times N} \approx SR_V \times (1 + \frac{SR_H}{N}),$$

where *N* is the current block size, and SR_H (=2P_H) and SR_V (=2*Pv*) are the search range in the horizontal and vertical directions, respectively. In Level A scheme, because only *N* × *(N-1)* reference pixels are reused, the on-chip memory size is only *N* × *(N-1)* reference pixels. Level A scheme can totally reuse the overlapped region between two reference blocks of two successive search candidates in the horizontal direction.

Level B scheme improves the data reuse of search region in the Level A scheme. Level B scheme presents the data reuse of the SR-level loop in the horizontal and vertical directions, so Level B scheme can not only totally reuse the overlapped region in the horizontal direction but also reuse the overlapped region in the vertical direction, as shown in Figure 15(b). For one current block, the search range is inputted once.

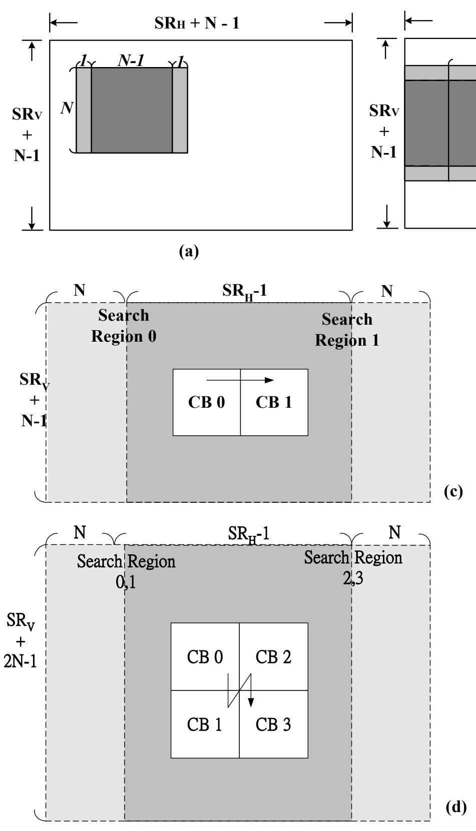


Figure 15. Data reuse schemes (a) Level A scheme (b) Level B scheme (c) Level C scheme (d) Level C+ Scheme for FSBMA, where the heavy gray region is the overlapped and reused region

Level A and Level B schemes reuse the overlapped region between the reference block of the successive search candidates in the SR-level loop. However, this kind of data reuse schemes still requires a lot of memory bandwidth, and may not be enough for a practical system. Therefore, Level C scheme is discussed to reuse the overlapped search region between two successive current blocks. Level C scheme is similar to Level A scheme but for the data reuse of the MB-level loop. As shown in Figure 15(c), there is a large overlapped region between two search regions of two successive current blocks in the horizontal direction. For two horizontal successive current blocks, only $N \times (SR_V+N-1)$ reference pixels are different and required to be updated.

An extension of Level C is possible, and it is named Level C+ scheme. By using stripe scan, not only the overlapped search region in the horizontal direction can be fully reused, but also the overlapped search region in the vertical direction can be partially reused, as shown in Figure 15(d). That is, several successive current MBs in the vertical direction are stitched, and the search region of these current MBs is loaded, simultaneously. Thus, only $N \times (SR_V+nN-1)$ pixels are required to be loaded from external memory if n successive vertical current MBs are stitched together, which is called n-stitched MBs.

Compared to the relationship between Level C and Level A schemes, Level D scheme is also similar to Level B scheme but for the MB-level loop. Level D scheme focuses on the data reuse of MB-level loop, so Level D scheme is the ultimate data reuse scheme of search region for one frame, which can fully reuse the overlapped search region not only in the horizontal direction but also in the vertical direction. However, because Level D scheme reuses the overlapped search region in both directions, the required on-chip memory size is very large, $(SR_V - 1) \times (SR_H + W - 1)$ pixels. The redundancy access factor, $Ra_{Level\ D}$, is 1.

Through the discussions above, we see the design trade-off between the required memory access and the on-chip memory size. Table 1 summarizes the bandwidth and on-chip memory requirements of these reuse schemes, and Table 2 shows the comparison of different data reuse schemes when the block size is 16 × 16, the search range is (-64,64) in both directions, and the frame format is D1 30fps. In order to reduce the intensive memory access for ME operations, the larger the on-chip memory size is required.

Table 1. The comparison of different data reuse schemes

	External Memory Bandwidth of Reference Frame (data access/pixel)	On-chip memory Size
Level A	$SR_V \times (1 + SR_H / N)$	$N \times (N - 1)$
Level B	$(1 + SR_V / N) \times (1 + SR_H / N)$	$(SR_H + N - 1) \times (N - 1)$
Level C	$1 + SR_V / N$	$(SR_H + N - 1) \times (SR_V + N - 1)$
Level C+	$1 + SR_V / nN$	$(SR_H + N - 1) \times (SR_V + nN - 1)$
Level D	1	$(SR_H + W - 1) \times (SR_V - 1)$

n is the number of stitched vertical current blocks.

Table 2. The comparison of different data reuse schemes for ME (the block size is 16×16, the search range is (-64,64) in both directions, and the frame format is D1 30fps)

	External Memory Bandwidth of Reference Frame		On-chip Memory Size
Reuse scheme	(Data access/pixel)	(MB/sec)	(pixels)
Level A	1,152	11,943.9	240
Level B	81	839.8	2,145
Level C	9	93.3	20,499
Level C+ (n=2)	5	51.8	22,737
Level D	1	10.4	107,569

4.3 Transform Coding

4.3.1 DCT

The 8×8 2-D DCT/IDCT is a widely used transform kernel in both image and video coding. It is also a computation-intensive module, and there are plenty of DCT/IDCT architectures in the literature.

Most DCT/IDCT architectures are based on fast algorithms instead of a direct mapping of Eqs. (1)–(2), each requires 4,096 (8^4) multiply-accumulate operations. Since DCT is a separable transform, row-column decomposition based architecture (Figure 16) is a commonly used architecture. The design in [39] is a good example to illustrate a complete design and optimization process of a row-column based DCT/IDCT processor. By row-column decomposition and the symmetry property, the number of multiplies is reduced to 512 (8^3). Therefore, the 1-D DCT/IDCT unit in the row-column decomposition architecture [39] needs to compute only eight multiplies per input sample with some simple changes of data sequences. This example again shows us that algorithm optimization before architecture mapping is significant and crucial.

The architecture optimization process usually can be partitioned into PE part, memory part and control part. For the PE optimization, in this case study, the architecture is compact since the horizontal and vertical 1-D DCTs are folded on one 1-D DCT unit. Also, the multipliers are hardwired ones since the DCT coefficients are constant. Furthermore, the signed digit representation of the DCT coefficients is adopted. This technique reduces the nonzero bits, and hence minimizes the number of adders required for the implementation of a hardwired multiplier. At last, finite wordlength simulation is necessary to decide the required wordlength to guarantee the computation precision. Figure 17 shows the row-column decomposition based 2-D DCT architecture.

The memory issue in a DCT/IDCT design is not very significant. In row-column decomposition architecture, a 64 words transpose memory is required. Since it is not a big memory, the optimization issue of this memory is minor. As for the control, the DCT/IDCT can operate seamlessly with the

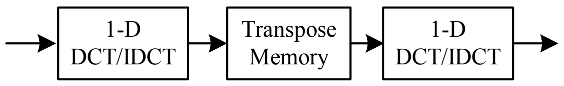


Figure 16. Row-column decomposition

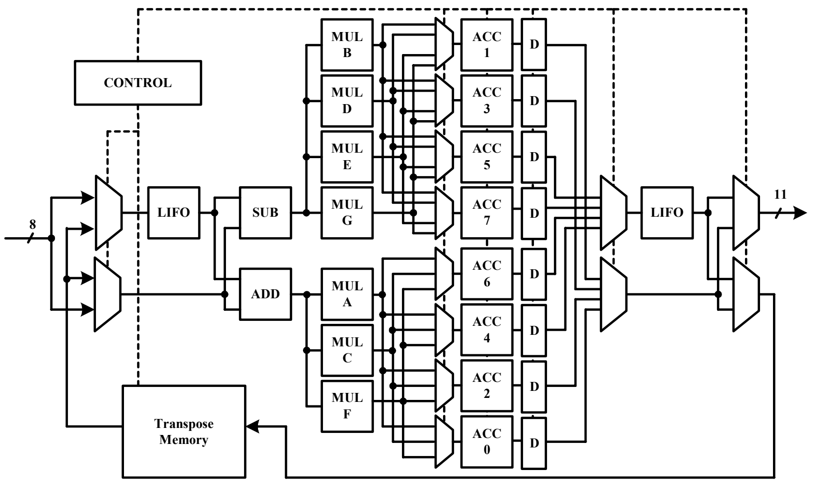


Figure 17. Row-column decomposition based 2-D DCT [39]

ability of one data per cycle of both the input data rate and the throughput rate. Therefore, the control signals are quite easy, and the I/O interface of the DCT/IDCT IP core is clear.

There are many other alternatives [41] for DCT/IDCT architecture. Besides the parallel multiplier-based design discussed above, architectures such as Distributed Arithmetic (DA) based, and digit-/bit-serial based ones, etc, are also widely discussed. Also, the direct 2-D architecture is another alternative for higher throughput. The design trade-offs are mainly among dimensions of the throughput, area, and control complexity, etc.

4.3.2 DWT

The design of a DWT consists of two parts, the 1-D PE design and the 2-D dataflow and scheduling. For the 1-D PE design, convolution-based and lifting-based implementations are two common approaches. Different approaches result in different numbers of multipliers and adders required. For the 2-D DWT, how 1-D DWT units are integrated and scheduled to perform 2-D DWT is a key that makes more significant architecture differences.

A direct mapping of the dataflow in Figure 3 is not efficient since the hardware cost is high and the hardware utilization is low. Usually, folded architecture is adopted. Different decomposition levels or horizontal and vertical filtering are mapped onto the shared PEs. Similar to the row-column decomposition 2-D DCT, a transpose buffer is required between the two 1-D DWT modules. However, the transpose buffer in DWT is a big issue, since the size of the buffer required for the transpose purpose in a DWT is usually much larger than that of a DCT. Take JPEG 2000 as example, typical tile sizes for DWT are 128×128 or 256×256, which is much larger than a small 8×8 DCT block. In the following, three design alternatives [43] are discussed: direct 2-D, line-based and block-based architectures.

Direct 2-D architecture allocates one set of low-pass and high-pass filters (Figure 18). The PE performs one direction's 1-D filtering first, then another direction's filtering, and the process continues for the next decomposition levels, if any. The characteristic of this architecture is that only single 1D DWT PE is required, and all the data accesses from the tile memory. That is, the tile memory is also used for the transpose purpose, and no extra buffer is required for intermediate results. If the tile memory is an off-chip memory, the intensive data access of the external memory will be the main concern.

For line-based architecture (Figure 19), there are two 1-D DWT PEs, one for the vertical 1-D DWT and the other for the horizontal 1-D DWT. Several lines of buffers are required for the transpose function, since this architecture schedules the second direction's filtering to start as early as possible when the filtering of the first direction has generated enough coefficients. How many lines are required for a line-based DWT architecture depends on the number of filter taps, and the length of a line depends on the image (a tile in JPEG 2000) width. The size of line buffers is much less than that of a whole tile, and are more feasible to be on-chip ones. The data access of the off-chip title memory will be less at the cost of one more 1-D DWT PE and several line buffers. This cost is usually worth paying since the bandwidth problem is usually more critical, and the power consumption of external memory access is much larger than the on-chip access.

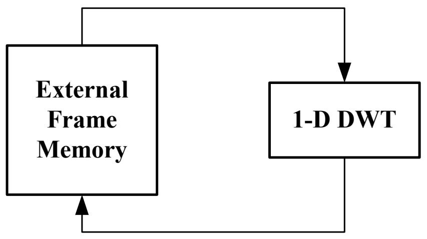


Figure 18. Direct 2-D DWT

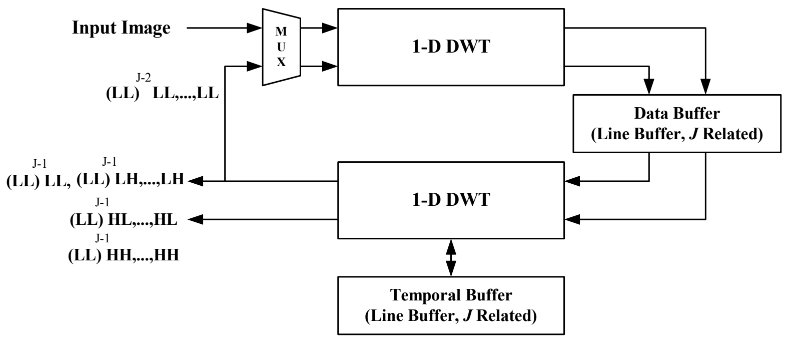

Input Image
MUX
1-D DWT
$(LL)^{J-2}$ LL,...,LL
Data Buffer (Line Buffer, *J* Related)
$(LL)^{J-1}$ LL, $(LL)^{J-1}$ LH,...,LH
$(LL)^{J-1}$ HL,...,HL
$(LL)^{J-1}$ HH,...,HH
1-D DWT
Temporal Buffer (Line Buffer, *J* Related)

Figure 19. Line-based multi-level 2-D DWT (J levels in this illustration)

Block-based 2-D DWT architecture is another alternative. The main consideration is to match the output data order of a DWT with the required input data order of the module after DWT. When a DWT process an image data in raster scan, the output will also be in raster scan order. However, in image coding, the coding flow of the module after the DWT may not be a raster way. In JPEG 2000, Embedded Block Coding with Optimized Truncation (EBCOT) is the module after DWT. The DWT coefficients of a tile are partitioned into non-overlapped code-blocks, which are the basic coding units of EBOCT. If the DWT output can be block-wise, then the buffer between DWT and EBCOT will be less compared with line-based architecture. The cost is that the input data order for the DWT has to be carefully controlled, and some extra buffer other than the size of a single code-block or repeated calculations will be necessary since the DWT filtering process requires some data across a code-block boundary.

4.4 Entropy Coder and Decoder

Different from the computation-intensive characteristic of the ME and DCT/IDCT, entropy coding consists of serial and more control-oriented bit-level operations. Therefore, an efficient entropy coder should have efficient bit-manipulation capability.

4.4.1 Huffman coder

Huffman encoding is a process to map fixed length symbols to variable length codewords, and the decoding is the inverse process. The throughput of tree-based VLC and VLD are low and not constant. For high speed image and video processing, parallel VLC and VLD architectures, which guarantee one symbol encoding and decoding per cycle, are better approaches.

The design of a VLC coder and a VLC decoder can be partitioned into two parts. One part is the table-lookup procedure, and the other part is the bit-manipulation for the variable-to-fixed concatenation in a VLC coder or the extraction of variable-length bits from the bitstream in a VLC decoder.

For a given Huffman table, the design of the table-lookup process in both VLC coder and VLC decoder is simple and straightforward. In most video coding standards, Huffman tables are usually fixed ones, so both the encoder and decoder know the Huffman tables in the beginning. In JPEG, user-customized Huffman tables are supported. Therefore, a general VLC decoder has to extract the Huffman table information from the JPEG bitstream in order to support the decoding of user-defined Huffman tables [44].

In a VLC coder, after the symbols are transformed to Huffman codewords, these variable length codes have to be packed into fixed length ones, said integer bytes. In a VLC decoder, the decoded bits have to be moved out from the bitstream, and the following bits have to be fetched for the next decoding process. The design in [45] provides good reference for high throughput VLC coder and decoder architectures.

4.4.2 Arithmetic coder

For the arithmetic coding in image and video standards, binary arithmetic coding is usually used. To further improve the coding performance, a context-based adaptive scheme is adopted. This characteristic of the context-based scheme is that context information requires some computation based on some previous coded data and some information has to be stored for later reference. Besides, due to the adaptive scheme, parallel architecture for arithmetic coding is difficult. The coding of a symbol depends on the updated probability, so symbols with some context cannot be parallel processed.

In standards, arithmetic coding is an iterative process of conditional branches and arithmetic, which are usually represented by a flowchart. With the well-defined data flow at hand, the first step of design process is simply to map the data flow directly into logic gates. This direct mapping is functionally correct but usually cannot meet the critical path constraint. Then, pipelining techniques are applied to shorten the critical path to meet the target operational frequency and throughput. The key optimization is to apply techniques to break some long paths to short ones, and also to conquer the loop operations by the technique similar to the design concept of the carry select adder.

Figure 20 shows a generic three-stage pipelined adaptive arithmetic encoder. There is a feedback loop between stage 0 and stage 1. Stage 1 updates the probability information based on the information from stage 0,

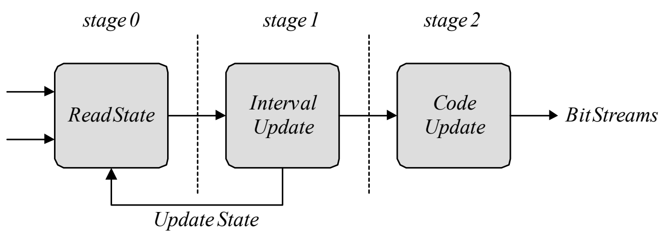


Figure 20. Generic three-stage pipelined context-based adaptive arithmetic coder architecture

and then passes the related updated information back to stage 0. When two successive symbols are with the same contexts, both the two possible branches are implemented, and the feed back information selects the correct branch result.

5. CODEC DESIGN: CASE STUDIES

To design an image or video codec system, detailed system analysis is the first key step and then the codec system can be divided and conquered by module-level design and optimization. Under the modulized design concept, once the key modules are available, the design of a codec is a process to integrate those modules. The key is to have a smooth data flow so that large amount of image/video data can flow through the processing pipeline as seamlessly as possible. In this case, the codec system can buffer less intermediate data among modules and reduce data access back and forth between processing elements and memories.

In image coding, the complexity of encoding and decoding are similar. In video coding, it is an asymmetric coding in the sense that the complexity of encoding is much higher than that of decoding due the ME process in the encoding side. The encoder design has more room for optimization. Also, among the multiple parameters, modes, and tools provided in a standard, an encoder can selectively implement some of them based on the application requirements and rate-distortion-complexity trade-offs. However, for a general decoder, it has to support all specifications and parameters defined in a profile and level in order to claim standard compliant.

Based on the discussions of some basic components in the previous section, the analysis and design of image and video codec IPs of JPEG, JPEG 2000 and H.264/AVC will be discussed in the following. Here, we will focus more on the system architecture rather than individual module designs.

5.1 Case Study 1: Baseline JPEG Codec

JPEG is widely used for digital image compression. As the population of DSC, JPEG codec becomes an important design. Figure 21 shows the simplified functional block diagram of JPEG. Fast JPEG encoding and decoding can be easily achieved in modern PC. However, in consumer products without such high frequency and powerful processor, JPEG encoding and decoding can be a tough job especially when the resolution of DSC keeps increasing and the request of continuous photo shooting.

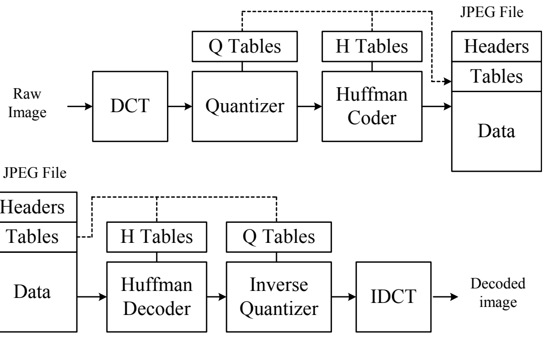


Figure 21. Simplified baseline JPEG functional block diagram

There have been many JPEG codec architecture proposed in the literature and in the market. Since there are no complicated loops between functional modules in JPEG, a fully pipelined JPEG architecture is feasible. A fully pipelined architecture feature is that it can encode or decode one pixel per cycle. That is, operated at *x* MHz, the encoder can process *x* Msamples per second, and the decoder can decode *x* Msamples per second.

In [42], a fully pipelined baseline JPEG encoder (JAGUAR architecture, Figure 22) is first presented. That paper presents a complete design from DCT to the data packer with detailed description of each module's architecture. That fully pipelined and modulized architecture sets a good example for reference. Each module can be replaced or modified with some tricks for improvement. For example, designers can choose any DCT architecture that is capable of one data input per cycle and one data output per cycle from many available alternatives.

Figure 23 shows a micrograph of a JPEG encoder prototype chip [44] for reference. It is implemented by TSMC 0.6μm 1P3M technology. The chip area is 5.38 x 5.35 mm^2. The encoder works up to 40 MHz. A fully pipelined baseline JPEG encoder does not require many memories. The minimum memory requirement is one for DCT transpose memory, one for zigzag

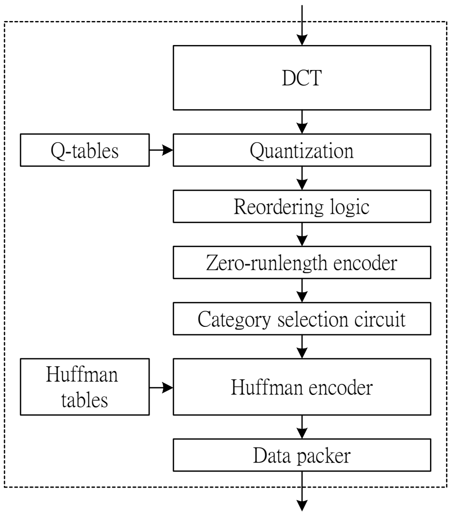


Figure 22. JAGUAR architecture [42]

reorder buffering, and one for quantization tables. The gate count of the JPEG encoder is around 33,000 logic gates [44]. The complexity of JPEG encoder and decoder are similar. A general JPEG decoder supporting user-defined Huffman table requires about 40,000 gates.

Besides the core part, the architecture can be extended more at the front end and the back end to have a completely stand-alone JPEG encoder IP. In that case, the stand-alone IP does not require any help from processors. In the front end, the color conversion of RGB to YcbCr can be considered. Also, since the basic coding unit of JPEG is an 8 × 8 block, a raster to block scan

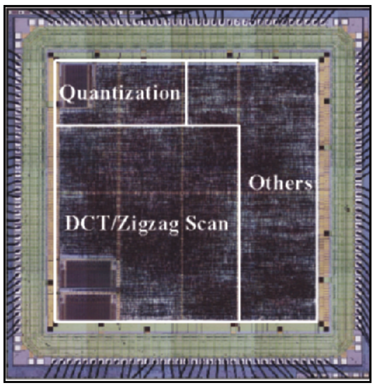


Figure 23. JPEG Encoder Chip micrograph [46]

conversion buffer is required. As for the back end processing after data packer, the packed data stream has to be checked and modified to avoid the ambiguous marker code due to data packing. Also, for applications that only require limited parameters for the user to select, only several fields in the bitstream header is variable. Therefore, a hardwired header generator can also be embedded. With these enhancements, the system processor can just provide an image in RGB format, and signal the JPEG encoder IP to start coding. Then, after processing, a standard compliant JPEG file is outputted by the JPEG IP. The processor is completely off-loaded from the JPEG processing.

5.2 Case Study 2: JPEG 2000 Codec

5.2.1 JPEG 2000 introduction

JPEG 2000 is the latest image coding standard. It is well known for its excellent coding efficiency as well as numerous useful features such as region of interest (ROI) coding and various types of scalability. Unlike JPEG, JPEG 2000 uses the Discrete Wavelet Transform (DWT) as the transformation algorithm and Embedded Block Coding with Optimized Truncation (EBCOT) as the entropy-coding algorithm. EBCOT can produce finely embedded bit streams that enable post-compression Rate-Distortion (R-D) optimization.

The design complexity of JPEG 2000 is much higher than JPEG. There are three critical issues to design a high throughput encoder. First, the DWT requires high memory bandwidth and enormous computational power. Second, the EBCOT requires extremely complicated control and sequential processing. Third, R-D optimization requires a large memory for storing the lossless code-stream and R-D information. All of the above requires high operating frequency, huge memory size, and high memory bandwidth for chip implementation.

5.2.2 81MS/s JPEG 2000 single-chip encoder with rate-distortion optimization [47]

We take the encoder design in [47] for the case study of JPEG 2000. The block diagram of the encoder is shown in Figure 24. The encoder consists of a main controller, a DWT module, a pre-compression R-D optimization controller, a parallel EBCOT module, and a dedicated Bit Stream Formatter (BSF). It is a tile-level pipelined architecture. Two 24 KB off-chip SRAMs are required. The input format is raw image data and the output is the JPEG 2000 code-stream.

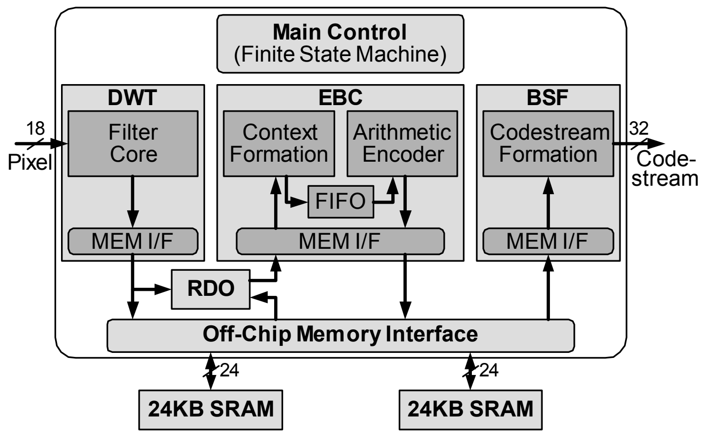


Figure 24. Block diagram of the JPEG 2000 encoder [47]

For the design of the DWT module, the recursive operations in the LL subband makes the data flow more complicated compared with DCT. Also, since a DWT tile is usually larger than a DCT 8 x 8 block, the memory issue is more problematic than DCT, as discussed in Section 4.3.2. The line-based DWT architecture is used in this design. The data buffer, which requires 1.5 lines of pixel data, stores the intermediate decomposition coefficients after the 1-D row DWT. The coefficients are then read by the 1-D column DWT to produce the 2-D results. The temporal buffer stores the intermediate data for the 1-D column DWT module, which requires 2 lines of pixel data for the (5,3) filter. Two 1-level 2-D line-based DWT modules are cascaded to implement the 2-level 2-D DWT decomposition and achieve a throughput of two pixels per cycle. Figure 25 shows the block diagram of the DWT module. The 1-D row and column DWT modules are implemented using a lifting scheme. 8,512 bits of on-chip memory, implemented by registers and on-chip SRAM, is required to accommodate the 128x128 tile size.

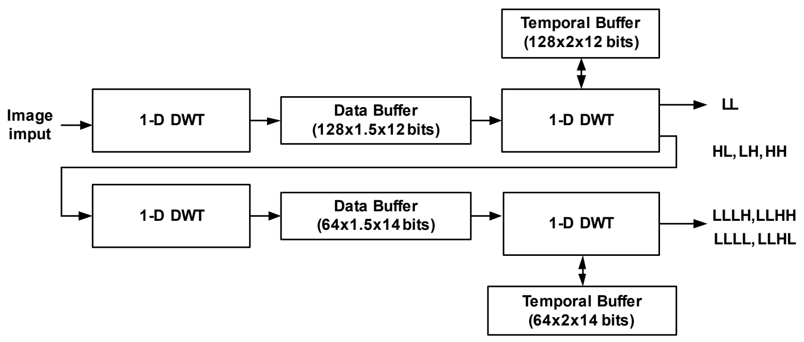


Figure 25. Block diagram of DWT module. [47]

The most critical design challenge of a high performance JPEG 2000 encoder is the design of a high throughput EBCOT module (Table 3, [48]). EBCOT is a bit-plane coder based on the context-based adaptive arithmetic coding. The operations are in bit-level. The equivalent input data rate is dramatically increased since a word (a DWT coefficient) now becomes many bits to be processed. The irregular fractional bit-plane coding order further complicates the data flow. Also, the interface between DWT and EBCOT is another issue, since there are two mismatches between the two modules. First, DWT is working in word-level while the EBCOT is working in bit-level. Second, the coding unit of DWT is a tile, while that of EBCOT is a code-block. Considerable buffer size and data access are therefore necessary if the two modules are directly connected.

Table 3. Run time profiling of JPEG 2000 encoding (The simulation is under JPEG 2000 VM 7.2 with image size 1792 x 1200, 5-level DWT and single layer, at PIII-733 PC.)

	Run Time Percentage (%)			
	Gray Scale Image		Color Image	
Operation	Lossless coding	Lossy coding	Lossless coding	Lossy coding
Color Transform	N.A.	N.A.	0.91	14.12
DWT	10.81	26.38	11.90	23.97
Quantization	N.A.	6.42	N.A.	5.04
EBCOT Tier-1	71.63	52.26	69.29	43.85
Pass 1	*14.89*	*14.82*	*13.90*	*12.39*
Pass 2	*10.85*	*7.00*	*10.94*	*5.63*
Pass 3	*26.14*	*16.09*	*25.12*	*13.77*
Arithmetic coding	19.75	14.35	19.33	12.06
EBCOT Tier-2	17.56	14.95	17.90	13.01

In EBCOT sequential coding mode, there is only limited parallelism for speed-up. What ASIC designers can do is to do a parallel check to decide which pass a bit should belong to. Sample skipping, group-of-column skipping, and pass-skipping schemes [48] are some of the techniques that can hide some of the bubble cycles in a sequential implementation. However, there is a fundamental limit of the throughput enhancement by these schemes. More parallelism in different levels should be explored for high speed encoding and decoding requirements such as motion JPEG 2000.

In this work, the EBCOT is operated in parallel mode. The proposed architecture can process a DWT coefficient in parallel, regardless of bit-width. Three techniques are applied to achieve this feature. First, a parallel context modeling approach instead of traditional bit plane-by-bit plane, is taken to increase the processing speed. Second, a reconfigurable FIFO (First-In First-Out) architecture that reduces bubble cycles is obtained by exploiting the features of the EBCOT and the DWT. Third, a folded Arithmetic Encoder (AE) architecture is devised to reduce the area.

The parallel EBCOT architecture is shown in Figure 26. It is capable of processing one 11-bit DWT coefficient per cycle. This architecture can process 28 passes in parallel, and therefore can operate at 1/28th the frequency of a traditional architecture. The state variables for context formation are calculated on the fly for each bit plane of each coefficient, so the 16kb state variable memory is no longer necessary. The folding technique reduces the hardware cost of the AE by 99 K gates.

The hardware implementation of the rate control function is also addressed in this design. For lossy coding, there are two drawbacks of the recommended post-compression R-D optimization algorithm in the reference software. First, the computational power and the processing time are wasted since the source image must be losslessly coded regardless of the target bit rate. Second, a large temporary memory is required to buffer the bit stream and side information for rate control. A pre-compression R-D optimization algorithm is proposed to solve these problems. The idea is to estimate the rate information of a pass before it is arithmetic coded. With a good model, the R-D optimization point can be approximated before EBCOT coding.

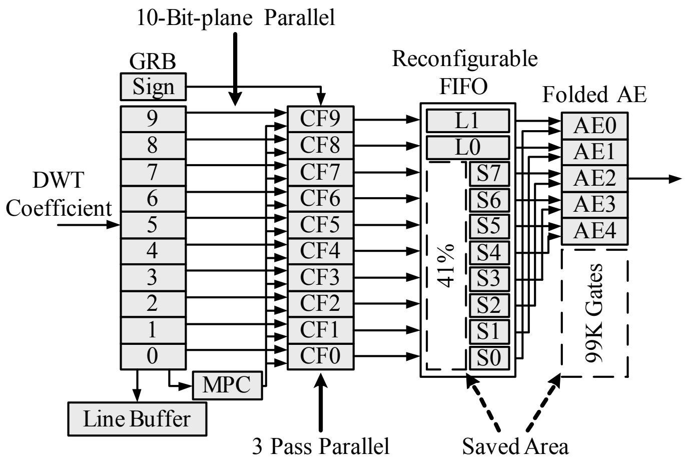


Figure 26. Word-parallel EBCOT architecture [47]

The flowchart of the proposed pre-compression R-D optimization algorithm is shown in Figure 27. It is comprised of two stages: accumulation and decision. During accumulation the distortion and bit-count are calculated and accumulated. In the decision stage the truncation points are determined according to the normalized distortion and estimated bit rate. The proposed algorithm allows the truncation point of a code-block to be determined before EBCOT encoding. Hence, only required coding passes are processed

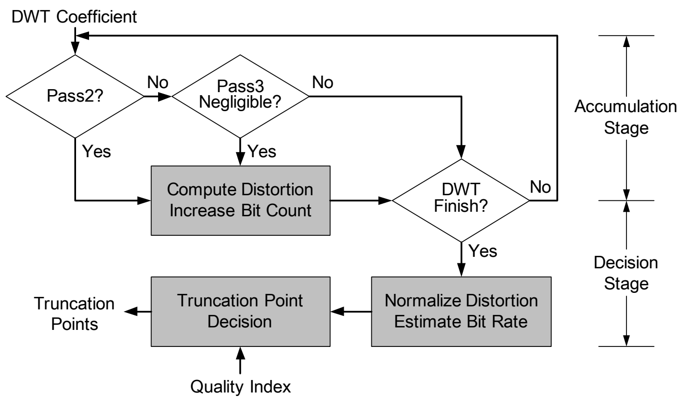


Figure 27. Flow chart of the pre-compression rate-distortion optimization algorithm [47]

which reduces the computational power as the compression ratio increases (e.g., compression ratio of 8 requires 8 times less EBCOT computation). In addition, the memory for lossless code-stream and R-D information is eliminated. The performance of the proposed pre-compression algorithm only degrades 0.3 dB on average compared with the post-compression algorithm.

Figure 28 is the micrograph of the 81 M samples/sec JPEG 2000 single chip encoder, which is implemented on a 5.5mm^2 die in 0.25μm CMOS

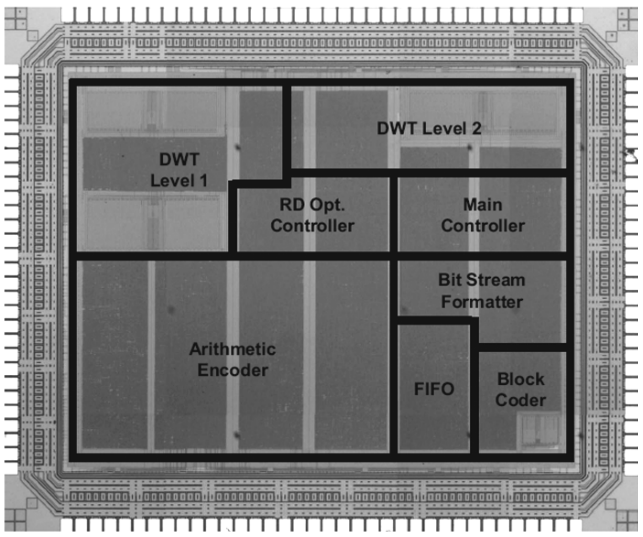


Figure 28. Die micrograph [47]

technology. The chip contains 163k gates and 11kb of SRAM. The processor consumes 348mW @ 2.5V when operating at 81 MHz. The detailed chip features are shown in Table 4.

Table 4. Chip specification

Technology	TSMC 0.25-μm 1P5M CMOS
Supply voltage	2.8V
Core area	$2.73 \times 2.02\text{mm}^2$
Logic gates	162.5 K (2-input NAND gate)
SRAM	7 K bits
Operating frequency	81 MHz
Power	348 mW
Package	PGA 256
Image size	Up to 32K $\times$ 32K
Processing rate	81 M samples/sec
DWT	(5,3) filter, 2-level decomposition
Tile size	128×128
Code-block size	64×64

5.2.3 Exploration of parallelism

Many JPEG 2000 codec designs are proposed to speed up the processing. The common idea among them is to explore more feasible parallelism at different levels, especially in the critical EBCOT design. Basically, according to the EBCOT algorithm, parallelism can be considered in several hierarchical levels: code-block level, bit plane level, pass level, symbol level, etc.

The previous case study is an extreme bit-plane parallel architecture, in which all bits of a coefficient are processed in parallel. One can view this block-coding engine as a word-level processor. The original word-level to bit-level mismatch between DWT and EBC is resolved. Therefore, no extra word-to-bit conversion and buffer is required. Also, since bits are processed in parallel, the state variable information can be processed on the fly, and therefore the state variable memories are not necessary.

For other alternatives, in [49] (Figure 29), a 2-plane parallel, 3-pass parallel, and 4-symbol parallel EBCOT architecture are implemented. The three-code-block parallel architecture [50][51] (Figure 30) is also a common solution. It tries to balance the throughput and input data rate between DWT and EBCOT. A double-encoder architecture is proposed in [52] to achieve real-time HD-movie encoding.

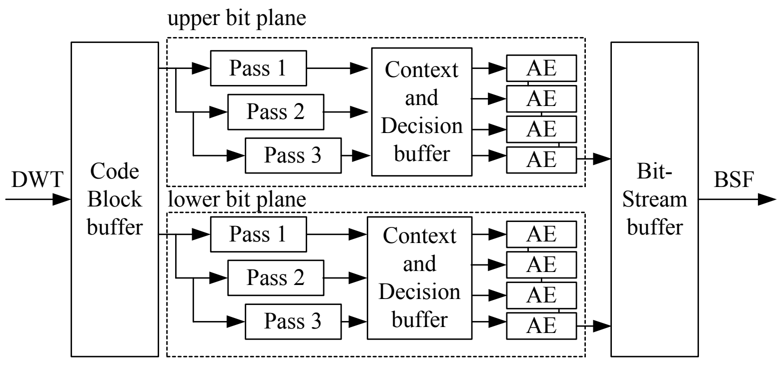


Figure 29. EBCOT parallel architecture [49]

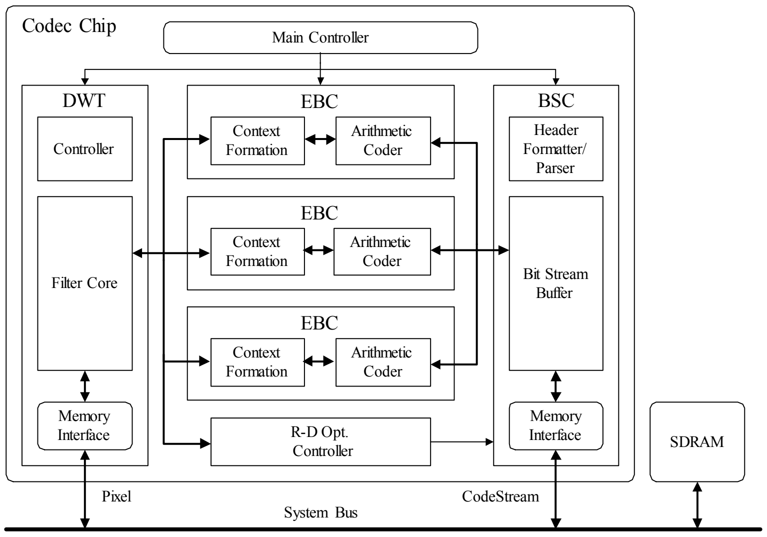


Figure 30. JPEG 2000 codec based on three-code-block parallel architecture

5.3 Case Study 3: H.264/AVC Codec

5.3.1 H.264/AVC introduction

Figure 31 shows the functional block diagram of the H.264/AVC [53][54][55] and highlights the features of some modules. Compared to MPEG-4, H.263, and MPEG-2, H.264/AVC can achieve 39%, 49%, and

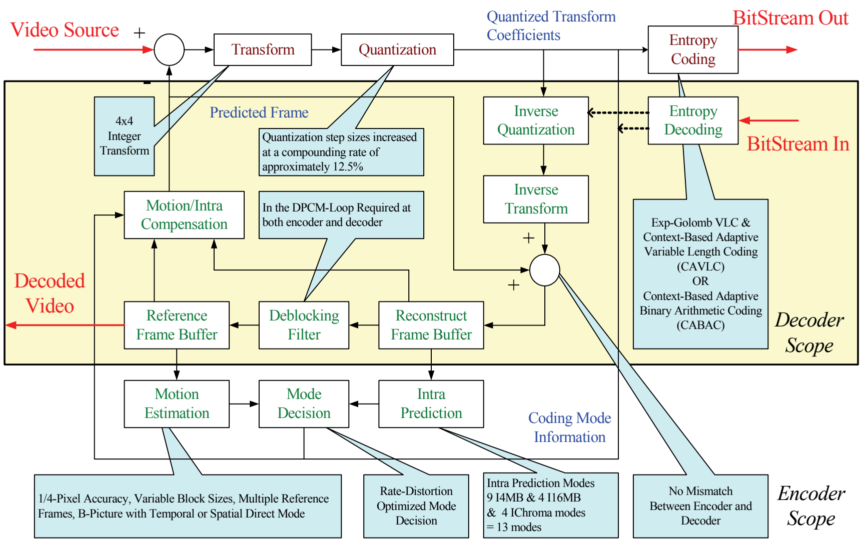

Video Source
+
-
Transform
Quantization
Quantized Transform Coefficients
Entropy Coding
BitStream Out
4x4 Integer Transform
Predicted Frame
Quantization step sizes increased at a compounding rate of approximately 12.5%
Inverse Quantization
Entropy Decoding
BitStream In
Motion/Intra Compensation
In the DPCM-Loop Required at both encoder and decoder
Inverse Transform
+
+
Exp-Golomb VLC & Context-Based Adaptive Variable Length Coding (CAVLC) OR Context-Based Adaptive Binary Arithmetic Coding (CABAC)
Decoded Video
Reference Frame Buffer
Deblocking Filter
Reconstruct Frame Buffer
Decoder Scope
Motion Estimation
Mode Decision
Intra Prediction
Coding Mode Information
1/4-Pixel Accuracy, Variable Block Sizes, Multiple Reference Frames, B-Picture with Temporal or Spatial Direct Mode
Rate-Distortion Optimized Mode Decision
Intra Prediction Modes 9 I4MB & 4 I16MB & 4 IChroma modes = 13 modes
No Mismatch Between Encoder and Decoder
Encoder Scope

Figure 31. Block diagram of H.264/AVC video coding system

64% of bit-rate reduction, respectively [56]. The high compression performance comes mainly from the new prediction techniques used to efficiently remove temporal and spatial redundancies. Intra prediction, unlike the previous standards, is done in spatial domain and has multiple modes. Inter prediction is enhanced by ME with quarter-pixel accuracy, variable block sizes (VBS), multiple reference frames (MRF), weighted bi-prediction and improved spatial/temporal direct mode. Moreover, the advanced entropy coding tools use content adaptivity to further reduce statistic redundancy. The perceptual quality is improved by in-loop de-blocking filter. Meanwhile there is no mismatch between decoder and encoder for integer transform scheme.

5.3.2 System analysis

The coding performance of H.264/AVC comes at the price of huge computational complexity. According to the instruction profiling with HDTV specification, H.264/AVC decoding process requires the computation of 83 Giga-instructions per second (GIPS) and memory access requirement of 70 Giga-bytes per second. As for H.264/AVC encoder, up to 3.6 Tera-instructions per second (TIPS) and 5.6 Tera-bytes per second computational resources are required. Dedicated hardware is a must to make most of H.264/AVC applications feasible.

It is a tough job to map the H.264/AVC procedures into the efficient system architecture. In addition to extraordinary huge computational complexity and memory access requirement, the coding path including prediction, reconstruction, and entropy coding is very long. The involved functionalities are not only abundant but also complex. Therefore, an efficient task partition with pipelining structure is required. Besides, an efficient memory hierarchy with data reuse scheme is essential to reduce the bandwidth requirement.

Furthermore, the architecture design for the significant modules is also very challenging. Figure 32 and 33 shows the run time profile of H.264/AVC encoding and decoding, respectively. The inter prediction takes 97.32% of the computational load, and obviously is the processing bottleneck of an H.264/AVC encoder. For a decoder, the inter prediction and de-blocking filter contribute the most computation time (39% and 36%), while IQ/IDCT, entropy decoding, and intra prediction occupy the rest.

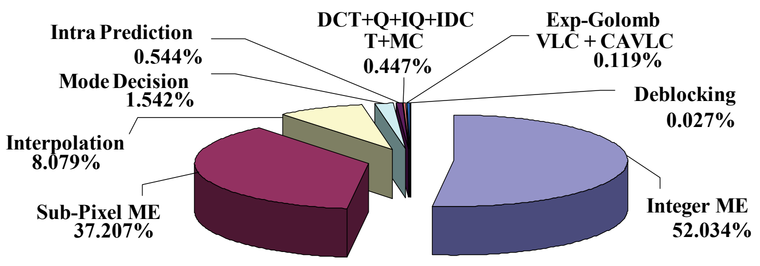


Figure 32. Run time profile of H.264/AVC inter frame coding

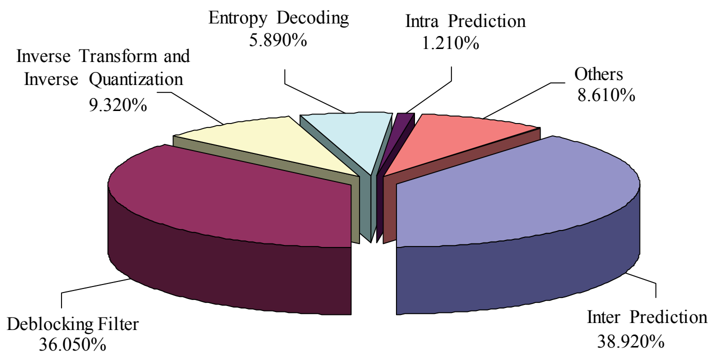


Figure 33. Run time profile of H.264/AVC decoding

The reference software adopts many sequential processing of each block in the macroblock (MB), which restricts the parallel processing. The coding tools involve with many data dependencies to enhance the coding performance, but the considerable storage space is the penalty. The block-level reconstruction loop caused by intra prediction will induce the bubble cycles and decrease the hardware utilization and throughput. Last but not least, there are functionalities that have multiplex modes, and the re-configurable engine to achieve resource sharing is a key for efficient implementation.

5.3.3 Architecture design

The encoder design in [57] and the decoder design in [58] are chosen for our case study of H.264/AVC codec.

For the encoder part, the traditional two-stage MB pipeline, prediction (ME) and block engine (MC+DCT+Q+IQ+IDCT+VLC), is not suitable because of the long critical path and feedback loop. Figure 34 shows the four-stage macroblock pipelining architecture of the encoder. According to the analysis in [57], five major functions are extracted and mapped into four-stage MB pipelining structure with dedicated task scheduling. As for the decoder, Figure 35 shows the hybrid task pipelining architecture for the decoder. A hybrid task pipelining scheme, a balanced schedule with block-level, MB-level, and frame-level pipelining, is proposed to greatly reduce the internal memory size and bandwidth.

Moreover, the design consideration and optimization for its significant modules including bandwidth optimized motion compensation (MC) engine,

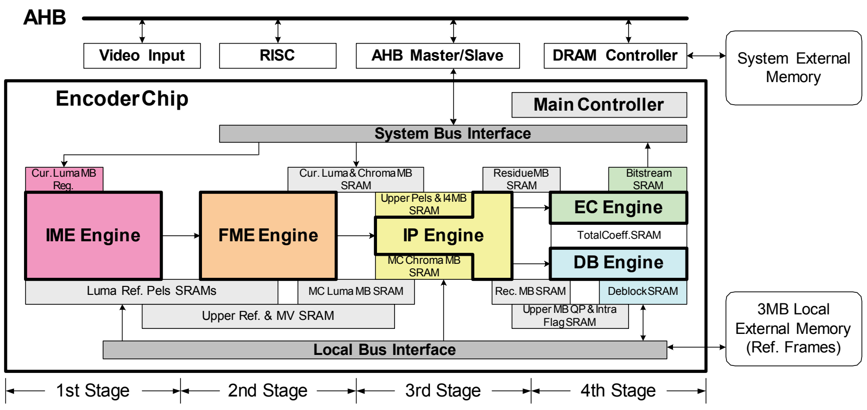


Figure 34. Block diagram of the H.264/AVC encoding system [57]

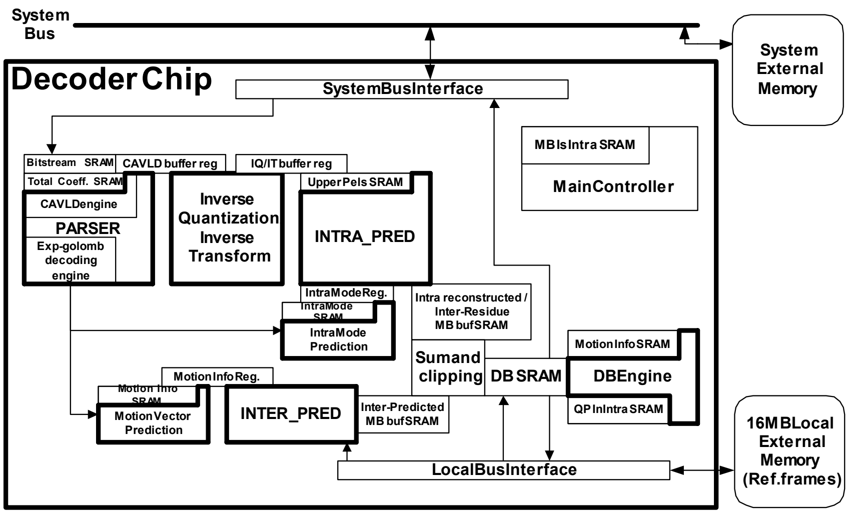


Figure 35. Block diagram of the H.264/AVC decoding system [58]

re-configurable intra predictor generator, parallel integer ME (IME) and fractional ME (FME) architectures are involved. The design shows that, by combining these efficient architecture and bandwidth reduction scheme, efficient implementation for H.264/AVC video coding system is achievable.

5.3.4 Prototype implementation

Detailed implementation data of both the encoder and decoder are provided here for reference.

5.3.4.1 H.264/AVC encoder

The encoder targets the baseline profile up to level 3.1. The maximum computational capability is real time encoding of SDTV 30fps with four reference frames or HDTV 30fps with one reference frame. The maximum processing capability is 108K MB/sec. This specification has 3.604 TOPS of computational complexity and 5.66 TByte/sec memory access requirement according to the profiling of the reference software implementation without any simplification.

Table 5 shows the logic gate count profile synthesized at 120 MHz. The total logic gate count is about 923K. Similar to the instruction profile, the prediction engine, including IME, FME, and INTRA stages, dominates 90% of logic area. As for on-chip SRAM requirement, 46 memory blocks, totally 34.72 K Bytes, are required. A prototype chip is fabricated by UMC 0.18μm

1P6M CMOS process. Figure 36 shows the chip micrograph, and the chip specification is in Table 6.

Table 5. Gate count profile of the H.264/AVC encoder [57]

Functional block	Gate counts	Percentage
Central control	34,151	3.7 %
IME stage	305,211	33.08 %
FME stage	401,885	43.55 %
INTRA stage	121,012	13.11 %
EC stage	29,332	3.18 %
DB stage	20,152	2.18 %
RAM BIST	11,025	1.19 %
Total	922,768	100%

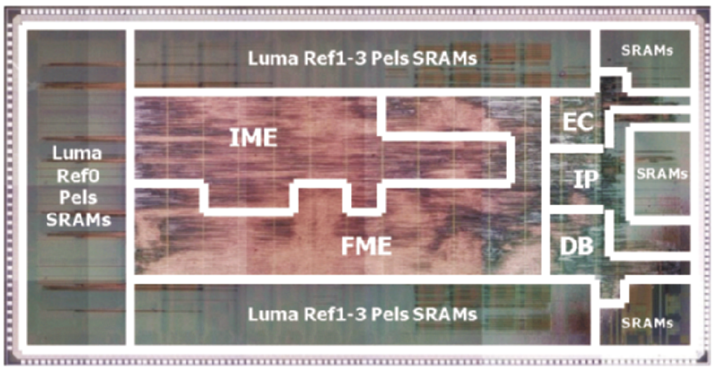


Figure 36. Chip micrograph of the H.264/AVC encoder [57]

Table 6. Chip specification of the H.264/AVC encoder [57]

Technology	UMC 0.18 μm 1P6M CMOS
Pad/Core voltage	1.8V
Core area	7.68×4.13 mm^2
Logic gates	922.8 K (2-input NAND gate)
SRAM	34.72 KByte
Encoding features	All baseline profile compression tools
Max. number of ref. frames	4
Max. search range (ref. 0)	H[-64,+63], V[-32,+31]
Max. search range (ref. 1-3)	H[-32,+31], V[-16,+15]
Operating frequency	81 MHz for D1 (4 ref. frames, Max. search range)
	108 MHz for HDTV 720p (1 ref. frame, Max. search range)
Power consumption	581 mW for D1
	785 mW for HDTV 720p

5.3.4.2 H.264/AVC decoder

The specification of this decoder is baseline profile at level 4.1. It can support real-time decoding of 2048 × 1024 video with 5 reference frames. The maximum operational frequency of this prototype chip is 120 MHz.

Table 7 shows the logic gate count profile. The total logic gate count is 217 K. 10 K bytes of on-chip SRAM are required. Figure 37 shows the layout view of the decoder. The core size is 2.19 × 2.19 mm^2. For highest specification, the power consumption is 186.4 mW for 2048 × 1024 30fps video format with 120 MHz operating frequency. For low power applications, the power consumption is 1.18 mW for QCIF (176 × 144) 15fps video format with 1.5 MHz operating frequency. The detailed chip specification is shown in Table 8.

Table 7. Gate count profile of the H.264/AVC decoder [58] (synthesized at 120 MHz operating frequency)

Functional block	Gate counts	Percentage
Central control	22,695	10.4 %
Entropy decoder	21,121	9.7 %
MC engine	69,695	32.1 %
Intra engine	28,707	13.2 %
IQ/IT	19,792	9.1 %
De-blocking filter	35,437	16.3 %
SRAM BIST	8,973	4.1 %
Misc.	11,043	5.1 %
Total	217,428	100 %

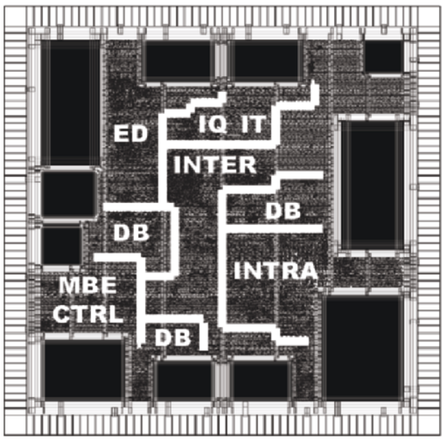


Figure 37. Chip layout of the H.264/AVC decoder [58]

Table 8. Chip specification of the H.264/AVC decoder [58]

Technology	TSMC 0.18 μm 1P6M CMOS
Pad/Core voltage	1.8V
Core area	2.19×2.19 mm^2
Logic gates	21.743 K (2-input NAND gate)
SRAM	9.98 KByte
Support features	All baseline profile compression tools
Maximum number of ref. Frames	5
Maximum search range	H[-2048,+2047], V[-512,+511]
Operating frequency	120 MHz for 2048×1024 30fps
Power consumption	186.4 mW for 2048×1024 30fps 1.18 mW for 176×144 15fps

6. SUMMARY

Multimedia IP development is one of the most important issues in a multimedia SOC design. In this chapter, an overview on how to design efficient image and video codecs are described. From theory to practice, the design methodologies and case studies are presented. Since the properties of high computation and high bandwidth requirement of image and video codecs, dedicated parallel hardware architecture can provide most powerful and efficient design. Even if a programmable solution is being considered, the know-how of dedicated architecture design will be the foundation for the programmable architecture to enhance its processing ability.

ACKNOWLEDGEMENTS

The authors would like to thank all students and alumni of DSP/IC Design Lab, National Taiwan University for their research contributions and help about the writing of this chapter. The authors also thank the National Science Council (NSC), R.O.C., the Ministry of Education (MOE), R.O.C., the MediaTek Foundation and the SiS Education Foundation for their support of research, and thank the National Chip Implementation Center (CIC) for their service and support of IC implementation.

REFERENCES

1. ISO/IEC, Int. Standard DIS 10918, "Digital compression and coding of continuous-tone still images."
2. ISO/IEC 15444-1:2000, "Information technology – JPEG 2000 image coding system – Part 1: Core coding system."

3. ITU-T Recommendation H.264 and ISO/IEC 14496-10 AVC, Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, 2003.
4. W. Kou, *Digital image compression algorithms and standards*, Norwell, MA: Kluwer Academic Publishers, 1995.
5. V. Bhaskaran and K. Konstantinides. *Image and Video Compression Standards: Algorithms and Architectures*. Norwell, MA: Kluwer Academic Publishers, 1997.
6. Y. Q. Shi, H. Sun, *Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms, and Standards*, CRC Press, 1999.
7. K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. New York: Academic, 1990.
8. W. P. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*. New York: Van Nostrand Reinhold, 1992.
9. D. S. Taubman and M. W. Marcellin, *JPEG2000 image compression fundamentals, standards and practice*. Norwell, MA: Kluwer Academic Publishers, 2002.
10. P. Kuhn, "Acomplexity analysis tool: iprof (ver 0.41)," ISO/IEC JTC/SC29/WG11, Dublin (Ireland), Doc. M3551, July 1998.
11. Iain E.G. Richardson, *Video codec design: developing image and video compression systems*, Chichester: Wiley, 2002.
12. F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappele, *Custom memory management methodology: exploration of memory organization for embedded multimedia system design*, Norwell, MA: Kluwer Academic Publishers, 1998.
13. Y.-W. Huang, C.-Y. Chen, C.-H. Tsai, C.-F. Shen, and L.-G. Chen, "Survey on Block Matching Motion Estimation Algorithms and Architectures with New Results" to Appear, *IEEE Journal of VLSI Signal Processing.*
14. Digital Video Coding Group, ITU-T recommendation H.263 software implementation, Telenor R&D, 1995.
15. W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Trans. Image Processing*, vol. 4, no. 1, pp. 105-107, Jan. 1995.
16. M. Bierling, "Displacement estimation by hierarchical block matching," *Proc. of SPIE Visual Commun. Image Processing (VCIP'88)*, 1988, pp. 942-951.
17. B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, no. 2, pp. 148-157, Apr. 1993.
18. Z. L. He, C. Y. Tsui, K. K. Chan, and M. L. Liou, "Low-power VLSI design for motion estimation using adaptive pixel truncation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, no. 5, pp. 669-678, Aug. 2000.
19. J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, no. 4, pp. 369-377, Aug. 1998.
20. S. Zhu and K. K. Ma, "A new diamond search algorithm for fast blockmatching motion estimation," *IEEE Trans. Image Processing*, vol. 9, no. 2, pp. 287-290, Feb. 2000.
21. A. M. Tourapis, O. C. Au, M. L. Liou, G. Shen, and I. Ahmad, "Optimizing the mpeg-4 encoder – advanced diamond zonal search," *Proc. of IEEE Int. Symp. Circuits Syst. (ISCAS'00)*, 2000, pp. 674-677.
22. A. M. Tourapis, O. C. Au, and M. L. Liu, "Highly efficient predictive zonal algorithms for fast block-matching motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 10, pp. 934-947, Oct. 2002.

23. T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated inter frame coding for video conferencing," *Proc. Nat. Telecom-mun. Conf.*, 1981, pp. C9.6.1-C9.6.5.
24. J. Jain and A. Jain, "Displacement measurement and its application in internal image coding," *IEEE Trans. Commun.*, vol. COM-29, no. 12, pp. 1799-1808, Dec. 1981.
25. M. J. Chen, L. G. Chen, and T. D. Chiueh, "One-dimensional full search motion estimation algorithm for video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, no. 5, pp. 504-509, June 1994.
26. R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, no. 4, pp. 438-442, Aug. 1994.
27. L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 3, pp. 313-317, June 1996.
28. L. K. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Trans. Circuits Syst.Video Technol.*, vol. 6, no. 4, pp. 419-422, Aug. 1996.
29. Y.W. Huang, S. Y. Ma, C. F. Shen, and L. G. Chen, "Predictive line search: an efficient motion estimation algorithm for mpeg-4 encoding systems on multimedia processors," *IEEE Trans. Circuits and Syst. Video Technol.*, vol. 13, no. 1, pp. 111-117, Jan. 2003.
30. D. Tzovaras, M. G. Strintzis, and H. Sahinolou, "Evaluation of multiresolution block matching techniques for motion and disparity estimation," *Signal Processing: Image Commun.*, vol. 6, pp. 56-67, 1994.
31. J. H. Lee, K. W. Lim, B. C. Song, and J. B. Ra, "A fast multi-resolution block matching algorithm and its VLSI architecture for low bit-rate video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 12, pp. 1289-1301, Dec. 2001.
32. K. M. Yang, M. T. Sun, and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," *IEEE Trans. Circuits Syst.*, vol. 36, no. 2, pp. 1317-1325, Oct. 1989.
33. T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Trans. Circuits Syst.*, vol. 36, no. 2, pp. 1301-1308, Oct. 1989.
34. Y. S. Jehng, L. G. Chen, and T. D. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms," *IEEE Trans. Signal Processing*, vol. 41, no. 2, pp. 889-900, Feb. 1993.
35. W.-M. Chao, C.-W. Hsu, Y.-C. Chang, and L.-G. Chen, "A novel motion estimator supporting diamond search and fast full search," *Proc. of IEEE Int. Symp. Circuits Syst. (ISCAS'02)*, 2002, pp. 492-495.
36. M.-Y. Hsu, "Scalable module-based architecture for MPEG-4 BMA motion estimation," M.S. thesis, National Taiwan Univ., June 2000.
37. J.-C. Tuan, T.-S. Chang, and C.-W. Jen, "On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," *IEEE Trans. CSVT*, vol. 12, no. 1, pp. 61-72, Jan. 2002.
38. C.-T. Huang, C.-Y. Chen, Y.-H. Chen, and L.-G. Chen, "Memory analysis of VLSI architecture for 5/3 and 1/3 motion-compensated temporal filtering," *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2005.
39. A. Madisetti and A. N. Willson, "A 100 MHz 2-D 8x8 DCT/IDCT processor for HDTV applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, no. 2, pp. 158-165, Apr. 1995.
40. W. Chen *et al.*, "A fast computational algorithm for the discrete cosine transform," *IEEE Trans. Commun.*, vol. COM-25, pp. 1004-1009, Sept. 1977.

41. G. S. Taylor and G. M. Blair, “Design for the discrete cosine transform in VLSI,” *IEE Proc. Comput. Digit. Tech.*, vol. 145, no. 2, pp. 127-133, Mar. 1998.
42. M. Kovac and N. Ranganathan, “JAGUAR: a full pipelined VLSI architecture for JPEG image compression standard,” *Proc.of the IEEE*, vol. 83, no. 2, pp. 247-258, Feb. 1995.
43. C.-T. Huang, P.-C.Tseng, L.-G. Chen, “Analysis and VLSI architecture for 1-D and 2-D discrete wavelet transform,” *IEEE Trans. Signal Processing*, vol. 53, no. 4, pp. 1575-1586, Apr. 2005.
44. C.-J. Lian, H.-C. Chang, K.-F. Chen, and L.-G. Chen, “A JPEG decoder IP Core supporting user-defined Huffman table decoding,” *Proc. of the 9th International Symposium on Integrated Circuits, Devices and Systems (ISIC-2001)*, Singapore, pp. 497-500, Sep. 2001.
45. S.-M. Lei and M.-T. Sun, “An entropy coding system for digital HDTV applications,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 1, no. 1, pp. 147-155, Mar. 1991.
46. C.-J. Lian, L.-G. Chen, H.-C. Chang, and Y.-C. Chang, “Design and implementation of JPEG encoder IP core,” *Proc. of Asia and South Pacific Design Automation Conf. (ASP-DAC 2001)*, Yokohama, Japan, pp. 29-30, Jan 2001.
47. H.-C. Fang, et al., “81MS/s JPEG2000 single-chip encoder with rate-distortion optimization,” *Digest of Technical Papers, 2004 IEEE International Solid-State Circuits Conference (ISSCC 2004)*, pp. 328-531 Vol.1
48. C.-J. Lian, K.-F. Chen, H.-H. Chen, and L.-G.Chen, “Analysis and architecture design of block-coding engine for EBCOT in JPEG 2000,” *IEEE Trans. Circuits and Syst. For Video Technol.*, vol. 13, no. 3, pp. 219-230, Mar. 2003.
49. H. Yamauchi, S. Okada, K. Taketa, T. Ohyama, Y. Matsuda, T. Mori, S. Okada, T. Watanabe, Y. Matsuo, Y. Yamada, T. Ichikawa, Y. Matsushita, “Image processor capable of block-noise-free JPEG2000 compression with 30 frames/s for digital camera applications,” *Digest of Technical Papers, 2003 IEEE International Solid-State Circuits Conference (ISSCC 2003)*, pp. 476-477 vol.1.
50. K. Andra, C. Chakrabarti, T. Acharya, “A high-performance JPEG2000 architecture,” *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 13, no. 3, pp. 209-218, Mar. 2003.
51. B.-F. Wu, C.-F. Lin, “An efficient architecture for JPEG2000 coprocessor,” *IEEE Trans. on Consumer Electronics*, vol. 50, no. 4, pp. 1183-1189, Nov. 2004.
52. H. Yamauchi, S. Okada, K. Taketa, Y. Matsuda, T. Mori, T. Watanabe, Y. Matsuo, Y. Matsushita, “1440 × 1080 pixel, 30 frames per second motion-JPEG 2000 codec for HD-movie transmission,” *IEEE Journal of Solid-State Circuits*, vol. 40, no. 1, pp. 331-341, Jan. 2005.
53. T. Wiegand, G. J. Sullivan, G. Bjntegaard, A. Luthra, “Overview of the H.264/AVC video coding standard,” *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560-576, Jul. 2003.
54. J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, T. Wedi, “Video coding with H.264/AVC: tools, performance, and complexity,” *IEEE Magazine on Circuits and Systems*, vol. 4, no. 1, pp. 7-28, first quarter, 2004.
55. A. Puri, X. Chen, and A. Luthra, “Video coding using the H.264/MPEG-4 AVC compression standard,” *Trans. on Signal Processing: Image Communication*, vol. 19, no. 9, pp. 793-849, Oct. 2004.
56. A. Joch, F. Kossentini, H. Schwarz, T. Wiegand, and G. J. Sullivan, “Performance comparison of video coding standards using Lagragian coder control,” *Proc. of 2002 International Conference on Image Processing (ICIP 2002)*, 2002, pp. 501-504.
57. Y.-W. Huang, T.-C. Chen, C.-H. Tsai, C.-Y. Chen, T.-W. Chen, C.-S. Chen, C.-F. Shen, S.-Y. Ma, T.-C. Wang, B.-Y. Hsieh, H.-C. Fang, L.-G. Chen, “A 1.3TOPS H.264/AVC

single-chip encoder for HDTV applications," *Digest of Technical Papers, 2005 IEEE International Solid-State Circuits Conference (ISSCC 2005),* 2005, pp. 128-130

58. T.-W. Chen, Y.-W. Huang, T.-C. Chen, Y.-H. Chen, C.-Y. Tsai, L.-G. Chen,, "Architecture design of H.264/AVC decoder with hybrid task pipelining for high definition videos," *Proc. of 2005 International Symposium on Circuits and Systems (ISCAS 2005)*, 2005, pp. 2931-2934.

Chapter 4

SoC MEMORY SYSTEM DESIGN

Kun-Bin Lee, and Tian-Sheuan Chang

MediaTek Inc.,5F, No.1-2, Innovation RD.1,Science-based Industrial Park, Hsin-chu City, Taiwan 300, R.O.C.
Dept. Electronics Engineering, National Chiao-Tung University, 1001 Ta-Hsueh Rd., Hsinchu, Taiwan

Abstract: As the increasing integration density of various IPs into the SoC, the memory system becomes a dominant role to determine the final performance, area, and power consumption of SoC. The memory system design involves various aspects, from bottom level on-chip or off-chip memory technologies, to the high level memory optimization and management. Between the two levels is the memory controller to efficiently deliver the required data within the power and delay constraints. The PC-driven off-chip memory continues its high density and high bandwidth development track. However, it also adapts its interface and power to be either fast random access or low power consumption to fit into the divergent needs of various SoC applications. The embedded memory now is driven by the SoC and thus becomes more integration friendly, either at the interface or at the process technologies. Memory optimization and management optimizes the memory access by high level reordering, remapping and memory size compression. Power of the memory system can be further reduced by transition reduction of memory bus and dynamic power management of memory systems. Further optimization of memory access needs the memory controller to fully utilize the available bandwidth. Since the components in SoC have divergent needs, either bandwidth sensitive, or latency sensitive, the memory controller design also quick evolves to be a more intelligent one to provide the different quality and latency guaranteed access. The optimization of memory system is part of the complex SoC design problem, which can only be analyzed and solved within the target applications

Keywords: SoC memory system, memory controller

Recently multimedia and wired/wireless communication technologies have fundamentally changed the way we create, communicate, and consume audiovisual information. These technologies have not only transformed existing applications and services like the distribution of entertainment video to the home but also spawned brand new industries and services like video-conferencing, direct-to-home satellite distribution, digital video recording, video-on-demand services, high-definition TV, video on mobile devices, streaming video, etc. Realization of these applications relies on VLSI for cost-effective implementation.

Researches on architecture design have shown that 50-80% of the area cost in (application-specific) architectures for real-time multi-dimensional signal processing is due to *memory components*, e.g., embedded SRAMs, and register files[1]. The International Technology Roadmap for Semiconductors (ITRS) also shows that memory already accounts for over 50 percent of a typical SoC, and will grow to 94% by the year 2014[2]. Also the power consumption is heavily dominated by memory access both in custom hardware[3] and in processors[4]. Table 1 shows the relative energy of different operations[3,5]. Data transfer and memory access operations consume much more power than a data-path operation. For example, fetching an operand from an off-chip memory for an addition operation consumes 33 times more power than the addition itself in case of a processor. Furthermore, memory access performance, including latency and bandwidth, could significant affect system performance[6,7,8]. This is especially true in high-performance, memory-intensive applications, such as those for multimedia processing. Obviously, a promising avenue for further optimization of SoC design under various design constraints must take data transfer and memory subsystems into consideration.

In this chapter, we will discuss and review the issues and status of SoC memory systems from three aspects, under layer memory technology, upper layer memory optimization and management, and memory controller. First, we will review the current status and development trend of memory technology, used inside the chips or for the off-chip memory, and how they affect the SoC design. The latest development of embedded memory focuses on the smaller area size and technology-friendly integration, whereas the off-chip memory continues its development on two tracks: high density and high bandwidth, or the ultra low power memory for portable SoC applications.

Table 1. Relative energy per operation at a 1.5V supply in 0.8μm CMOS technology

Operation	Relative energy/op
16b carry-select adder	1
16b multiplier	3.6
8×128×16 SRAM (read)	4.4
8×128×16 SRAM (write)	8.9
External I/O access	10
16b memory transfer	33

These new developments bring new transfer characteristics and also new access limitation to the SoC designer. At the second part, we will review the current status of several memory optimization techniques, including memory-aware techniques and low power optimization for memory systems, especially their applications on the memory and bandwidth demanding multimedia chips. The final part reviews the recent development of the memory controller design. The development of memory controller is evolved from a simple protocol translation role to the smart bandwidth provider with different quality and latency guaranteed scheduling.

1. MEMORY TECHNOLOGY

Today's consumer electronics products process massive amounts of data for audio and video and other applications. Thus it demands variety of memory devices and memory size to work together, from volatile DRAM for run-time storage and nonvolatile flash for massive data and program storage. As the devices become portable, specialized low power memory, such as mobile pseudo SRAM or mobile DRAM emerges to support these battery-powered applications. Besides, as the target application of the embedded system diverges, the memory requirements are also diverse: graphics chips favor streaming, while networking equipment favors fast random access latency of 40-byte (IP) packets or 53-byte (ATM) packets. If the memory size demand is small enough, several megabytes, embedded memory can provide another high bandwidth solution that off-chip memories cannot offer, which is a solution that current high bandwidth demanded graphics chip or video encoder adopts. Divergent needs mean divergent technologies and thus heterogeneous integrations. These divergent memories will be reviewed in this section, first on the external memory especially the DRAM system, and then on the embedded memories. Since many books and papers have covered the memory technologies, our focus will be the development trend and the integration issues of these memory technologies. The discussion will more bias to RAMs instead of ROM or flash since they store the time-varying data and thus significantly affect the overall area, speed and power.

1.1 Off-Chip Memories

Traditionally, DRAM roadmap is largely driven by the PC products. These mainstream PC-favored DRAMs constitute more than 90% of total production, achieving the lowest price per MB, and thus widely supported and used in other domains. The development trends of PC-based DRAMs focus on supporting higher bandwidth, as defined by JEDEC. Most of the modern DRAM development is based on SDRAM. A typical SDRAM accesses one data at the positive clock edge. To improve its throughput, DDR SDRAM (Double data rate SDRAM) enables two (*2n*) instead of one (*1n*) data prefetch for one bus cycle by adopting double clock edge data sampling. Thus, the data transfer rate of DDR is twice of that in SDRAM. Furthermore, the latest DDR2 SDRAM enables 4n-prefetch, in contrast to 2n-prefetch realised in DDR. It means, in fact, that at each memory bus cycle, DDR2 transmits 4 (instead of 2) bits of information from logical (internal) memory chip banks into I/O buffers using one data interface line. This makes DDR2 SDRAM be able to provide quadruple bandwidth than SDRAM with the same core operating frequency. DDR2 SDRAM also added the features of posted CAS and additive latency to prevent DRAM command bus conflict. To operate at the high frequency, DDR2 SDRAM adopts the on-die termination to avoid signal reflections induced delay along the signal bus lines. The overall operating voltage is also reduced from 2.5V to 1.8V to save the power consumption and allow higher frequency operations. Due to the lowest cost-per-bit advantage, SDRAM and DDR SDRAM have been widely used in the low and medium bandwidth SoC system. For high bandwidth applications, DDR2 SDRAM will be a suitable solution.

The new development of DRAM improves the burst bandwidth. However, the random access latency almost remains unchanged for years. This bandwidth favor improvement partly is due to the cost-reduction centric improvement of DRAM industry, and partly is due to of the inherent DRAM architecture: one row access can get parallel data from the sense amplifier by selecting different column address. Patterson[9] has concluded that "bandwidth is improved by at least the square of the improvement in latency". This imposes the design challenge to the SoC designer: how to efficiently take advantage of the burst access bandwidth while supporting the low latency activities.

Thus, how to select the proper DRAM types for the SoC systems depends on the application needs. For streaming applications like video and audio processing, conventional DRAM is quite suitable. However, for the latency sensitive applications like networking, low latency DRAMs can satisfy both the latency and bandwidth needs.

Specialized low latency DRAMs like RLDRAM (reduced latency DRAM), or FCRAM (fast cycle RAM) have been developed for such special networking purpose[10]. The low latency DRAM improves the random access time by pipelining the memory access into three-stage: the address decoder, the memory array, and the I/O buffer. In a typical DRAM, the DRAM must first decode the row address, find the location in the memory array and then read/write the data from/to the I/O buffer. Since all functions are in series, a conventional DRAM cannot start the next row address sequence before completing all three stages. By pipelining these three functions, FCRAM is able to start a new row address access following previous one. The result is a random cycle time of 20 to 30 ns for FCRAM compared with 60 to 70 ns for other types of DRAMs, such as DDR SDRAM.

For mobile applications, low power consumption becomes one of the major concerns. SRAM has been the choice since it does not have a leaking per-cell charge-storage capacitor that requires periodic refreshing and draws the standby current. However, as more and more functions are squeezed into the system, the required amount of code and data storage also increases exponentially. Thus, the cost-per-bit advantage of DRAM over SRAM becomes apparent. To address both needs, size and power, specialized low power memory like mobile DDR SDRAM or pseudo SRAM have been developed.

Mobile DDR SDRAM incorporates several low power functionalities into DDR SDRAM to reduce the standby current. First, the supply voltage is reduced to 1.8V from 2.5V, providing half of the power reduction. Second, it now can support partial array self refresh (PASR), only half or one-fourth, instead of whole DRAM array. Thus, only the array with the valid data will be refreshed to avoid unnecessary self refresh current. To further reduce the self refresh current, it also lengthens the refresh period for lower temperature, called temperature compensated self refresh (TCSR). Finally, when the DRAM is not used, e.g., the standby mode in the mobile devices, these low power DRAMs can enter into the deep power down mode to completely turn off the self-refresh operation.

Pseudo SRAM, as its name suggests, combines advantages of the low cost DRAM cell and the simple traditional SRAM interface, non-multiplexed address bus. To solve the power consumption of DRAM cells, low power techniques adopted in mobile DDR SDRAM are also applied to the pseudo SRAM designs. Furthermore, to work like traditional SRAM, non-multiplexed address bus provides high speed random access time, and the internal self-refresh logic eliminates the refresh needs of DRAM cell.

The integration of off-chip memories is divergent from system-on-board level to the system-in-package level. Traditional system-on-board level integration has its advantage of well-established practices for years, including chip testing, assembly and board testing. However, as the system is getting higher operating frequency and smaller former factor, system-in-package (SiP) approach has also attracted designer's attention[11]. SiP integrates the required bare die of logic, memory and analog parts together into the same package. Since different components can use its best suitable technology, the component performance will not be sacrificed and on-same-package connections have much lower capacitive loading and thus lower delay than off-chip communications. The SiP integration could be a 2-D flat or 3-D vertical integration or mixed 2-D/3D. The 2-D flat style places all dies on the flat silicon and connects each other. The 3-D vertical style stacks all dies for connection. Since many dies are integrated in a small area, the heat dissipation and chip testing has to be considered for the overall package. How to design an effective system chip using either SoC, or SiP needs detailed evaluations on the cost, technologies and functional partitioning for the optimal integration.

1.2 Embedded Memories

With the VLSI technology stepping into the deep submicron era, hundreds of millions of transistors can be integrated into a single chip. Though more logic can be integrated into the chip, however, most of the transistors go to the memory, ranging from half to estimated 90% of the silicon real estate, according to ITRS. Embedded memory brings a lot of benefits but may not be cheaper and naturally coexist with other logic on the same chip.

Embedded memory offers the major benefits of low power, high bandwidth, high speed, and enhanced flexibility and the side benefits of minimal board space, and increased reliability. The embedded memory can offer flexible customized granular size like 66KB for the target application without the power-of-two size restriction like 16MB in the external memory. When the memory is integrated into the chip, all the high loaded and limited off-chip interconnections are replaced by a small loaded and almost unlimited on-chip communications. Thus, the power consumption and performance are improved significantly. This wide bandwidth is especially useful and attractive for bandwidth demanding applications like video and graphics, a major user for large embedded memories[12,13].

However, how to integrate these embedded memories with other logic circuits requires tradeoff on process compatibility and performance. Logic and memory are fundamentally different technologies with sometimes-contradict demands, from the interconnection requirement to the cell structure[14]. Logic usually scatters across the chip in an irregular fashion and requires more metal layers, six or more, for interconnection, while memory array is more symmetrical and repetitive and requires far less interconnection than logic. On the other hand, standard memory process requires four or more polysilicon layers for floating gate of EPROM, EEPROM, and flash memory, the resisters of a 4T SRAM cell, and the stacked-capacitor's DRAM cells, while logic process usually uses one or two polysilicon layers, just for transistors. An extra metal or polysilicon layer adds cost, complexity, and fabrication time. In addition, standard logic transistors have thin oxides with low threshold for fast switching time while many memory technologies prefer the thick oxides and high threshold for small leakage current. A compromise for logic or DRAM process will make logic slower or memory high leakage current. Beyond the process, the fast switching logic introduces noise to the sensitive memory core. The flash memory might require high voltage for programming and erasing. All of these issues have to be considered and clarified for integration.

In general, the integration approach can be classified into two approaches: logic process compatible embedded memories and dedicated process embedded memories[15].

Commonly used logic process compatible embedded memories are ROM and 6T (six-transistor) SRAM. ROM cells are much smaller than RAM cells. Due to its read-only features, its application is also limited to permanent lookup table or programs. 6T SRAM uses cross coupled and regenerative structure to offer high speed but also result in lower density. Thus it is preferred used in frequently access, time-critical storage or small amount of storage, like caches or local buffers. Beyond speed, power is also a concern for large embedded SRAMs. Since dynamic power of SRAM is linearly proportional to its size, N-bank partition can reduce the power by N-fold. For deep submicron process like 130nm and smaller feature size process, the leakage current will become a source of power consumption, which can be reduced by changing the SRAM circuit design or turning off the power supply for SRAM blocks to suppress the leakage current[16].

Dedicated process embedded memories usually add the extra processing steps for memory fabrication into the logic process. The required steps depend on the targeted memory technologies and design.

For SRAM, since traditional 6T SRAM occupies large area, small area SRAMs cell like 1T SRAM[17] have been developed to offer DRAM like density, about 1/4 or 1/5 area of 6T SRAM. 1T SRAM uses one transistor with one plane capacitor to avoid a complex extra process to build the stack or trench capacitor in conventional DRAMs. The plane capacitor can be built by adding one extra mask into the common logic process, which is more logic process compatible but also less charge. The required refresh operation is transparent to the user. An internal refresh timer is included into the design to generate periodic refresh requests to the memory banks. The refresh operation will be delayed if any access is active.

Direct porting the dedicated DRAM macro into logic process will require a lot of extra masks and processing steps, about 13 to 20 steps. Embedded DRAM suppliers reduce the cost added to the logic process by using an array of DRAM technologies ranging from traditional stacked and trench capacitors, to simple logic compatible planar capacitors, shallow trench capacitors and MIM (Metal-Insulator-Metal) capacitors. On the horizon is the development of capacitor-less DRAMs[18]. The latest development of embedded DRAM tries to be more logic process friendly and explore the unique benefit of the embedded environment, unlimited pin access, to enhance the access speed. The capacitor-less DRAMs exploit the floating gate effect to store the charge in the devices's body. This eliminates any extra processing steps. The trench capacitor based embedded DRAM can bury the capacitor under the logic and thus requires no extra process after the capacitor is fabricated. The embedded DRAM such as NEC eDRAM[19] changes the embedded DRAM interface into the non-multiplexed address and achieves the single cycle SRAM-like fast random access. The conventional SDRAM access commands are also simplified to be SRAM-like access. The whole embedded DRAM works like SRAM with the DRAM cell. Thus, the only extra work to do is the refresh operation. To be logic process friendly, the capacitor structure is also changed from the conventional high temperature PIP (Poly-Insulator-Poly) in the DRAM process to the low temperature MIS (Metal-Insulator-Silicon) or MIM (Metal-Insulator-Metal) capacitor structure. Low temperature process whose temperature is below that used in a normal CMOS logic process reduces the required thermal cycling that would otherwise cause transistor performance degradation during the fabrication.

Embedded nonvolatile memories have attracted many applications in modern SoCs, ranging from industrial, consumer, networking, office automation, to smart cards and RFID tags. They provide nonvolatile storage for frequently read but rarely written configuration information. However, the embedded nonvolatile memory still adds the process complexity by the introduction of the floating gate for the charge storage and retention, high-voltage transistors, as well as designs for routing the required high voltages to the memory array; and the complexity of decoding, sensing, timing circuits, and algorithms stored in state machines. For SoC integration, the new development diverges from the commodity flash to logic process compatible design, like SST's split-gate (SuperFlash) cell design[20].

Disregarding the underlying process, the integration of embedded memories is still a problem since the large area blocks will impose large blockage areas that place-and-route tools must detour around to connect the I/O and core logic areas. To be SoC friendly integration and with the aid of multiple level of metal layers, the latest embedded memories allow active signals to pass over the macros, and thus the router can take the shortest path between the I/O and core areas. This helps layout designers maintain signal integrity, and easily optimize critical timing paths. To avoid the signal interferences with the capacitance values of the embedded memories, these embedded memory macros or the designers usually have to add one grounded metal layer over the macro to isolate the macro. Since the number of metal layers for the embedded memory is usually no more than four, the modern up to seven metal layer process still provides enough extra layers for routing.

With the integration into the chips, the memory architecture is also more flexible to be adaptive to the target function. Various techniques can be applied here, like extending the memory hierarchy, memory partitioning, and bandwidth optimization. For more details, readers can refer to the survey papers by Benini et al.[15].

2. MEMORY OPTIMIZATION AND MANAGEMENT

Advanced VLSI techniques make both density and speed of logic manufacture processing much improved. But advanced manufacture processing of memory improves more in density and less in speed. This makes memory system become the bottleneck of the whole system performance, especially for those memory systems mainly using off-chip DRAMs. To improve memory peak bandwidth, approaches such as wider data buses, higher frequencies, double data rates or packetized protocols (e.g., Rambus DRAM and SLDRAM) are used. Moreover, these approaches are independent of the applications. Another kind of approach to improve memory system performance is to hoist the utilization of available bandwidth. This approach usually takes either or both of the characteristics of the memory system and the memory access behavior of applications into consideration. When considering image processing applications, for example, their behaviors are usually modeled as operations, such as transforming and filtering, on data arrays. Due to the native massive data of arrays, these arrays are often stored in off-chip DRAMs. While DRAMs are cheaper, they are slower and have memory access penalties (e.g., page miss and bank miss) that reduce the utilization of memory bandwidth. Thus simple one-to-one mapping between the array variables and the memory leads to an inefficient design. A lot of proposals to improve bandwidth utilization are based on the loop manipulations by exploiting the regularity and locality of application's memory access behavior with the characteristics of memory system. These loop manipulations include loop interchange, reversal, skewing, splitting, merging, and padding, whereas the target memory systems cover caches, scratch-pad memories, and DRAMs.

2.1 Reordering and Remapping

In this section, we briefly review two schemes, reordering and remapping, for memory access improvement. In the former approach, memory access can be statically or dynamically reordered. Ayukawa[21] uses a dynamic access-sequence control scheme to enhance random-access performance of embedded multibank DRAM macro. This scheme hides the page-miss penalty by reordering the access, if possible. The same authors design an access optimizer for embedded DRAM[22]. The access optimizer uses inter-bank, non-blocking access scheme to decide the sequence of data from different masters to reduce the influence of miss penalty while the sequence from data for the same master is kept unchanged. McKee et al.[23] propose a hardware-assisted access reordering by exploit page-mode operation of DRAM for vector computations to maximize the efficiency of the system memory bus. They also extend their work for systems with multiprocessors[24] and Direct Rambus memory[25]. Panda et al.[26] incorporate EDO DRAM access model into high-level synthesis and use loop transform to statically reorder the memory access to obtain better memory bandwidth utilization. They also extend their work for SDRAM[27], which exploits two new features of SDRAM, burst mode access and multiple bank architecture, to gain performance improvement.

Unlike memory access reordering that changes the order of accessed data, remapping changes the positions of data but keeps the access order unchanged. This may save buffers and reduce design complexity needed in reordering approaches. However, in some cases, such as random access, we do need reordering schemes to get better memory bandwidth utilization. On the other hand, if the characteristics of memory access can be predetermined, remapping approaches are usually a better way to gain more efficient memory access. Some works in this area are briefly introduced here. Panda and Dutt[28] propose a tile-based memory mapping to lower power through reducing address bus activity. Tile-based memory mapping is also used to tailor some specific applications, such as MPEG-2 video decoding[29], for SDRAM to reduce page breaks. Gleerup[30] uses both a tile-based rendering algorithm and a round-robin manner for memory bank accessing to hide the page open/close operations and hence achieve high memory bandwidth utilization. Chang and Lin[31] allocate arrays to different banks of SDRAM by utilizing SDRAM's multi-bank characteristic, though their assumption to allocate one row of any array to a memory page may not be practical enough when the row size of an array is either too small or too large. Schmit and Thomas[32] have presented faster and less hardware techniques for generating addresses for multiple single-dimensional arrays that have certain layout and size relationships.

One point must be noted is that the DRAM access ownership of each functional unit in multi-core systems also has to be guarded for a reasonable duration. Otherwise, the advantages of remapping are diminished.

2.2 Transition Reduction of Memory Buses

As mentioned earlier, memory accesses take a significant part of power consumption. Therefore, techniques to reduce this part of power dissipation are appreciated. A lot of works are investigated on the topics of reducing memory traffic[33] or reducing memory bus activity. As for reducing memory bus activity[5,7], optimized mapping of data[28], scheduling of memory access[26], and bus coding schemes[34,35] are proposed. Except for bus coding schemes, most of these techniques are also helpful in performance improvement.

To reduce transition of memory buses, including data bus and address bus, coding techniques are widely investigated. Due to the high correlation between consecutive addresses, most coding techniques focus on address buses, especially for instruction address buses. Gray coding[36] was proposed to minimize the transitions on the instruction address bus, such that there is only one transition between two consecutive addressees. However this coding scheme does not work well for data buses because these buses are typically not sequential. As for data buses, the bus-invert coding[37] and its variants[38,39] are more applicable. In bus-invert coding, for example, the total number of transitions occurring between the newly arrived data and the present data on the bus is first calculated. If this number is more than half the number of bus wires, then the data is inverted and sent on the bus. Otherwise, the data is sent as is. The inversion of the bus is signaled through an extra bit line. This extra bit, however, is not acceptable in some systems. To avoid the use of extra signals, Mamidipaka et al.[40] propose adaptive schemes based on self-organizing lists to exploit the spatial and temporal locality of the addresses. This approach reduces the transition activity of up to 54% in data address busses and up to 59% in multiplexed address busses.

To employ bus coding techniques, several issues must be considered, i.e., latency, extra control signals, and power consumption overhead of the chosen technique. Because the protocols of commodity memory cannot be modified, the space of practical coding schemes for memory buses is limited.

2.3 Reduction in Memory Size

Most embedded systems have tight bounds on memory space. Various techniques, therefore, have been investigated to conquer this issue. Compression is one of the ways of reducing memory footprint. Note that, if used correctly, compression can also improve the execution cycles and reduce power consumption as it reduces the amount of data that needs to be accessed from the memory and needs to be communicated over the bus. Furthermore, the smaller size of memory also dissipates less power.

Code compressions are widely applied to various processor architectures, including RISC, DSP and VLIW, etc. The compression granularity also ranges extensively from an instruction, a cache line, a basic block, and a function. Most compression schemes strive to produce the smallest possible encoding of their inputs. Program compaction is stressed by an extra condition: the compacted representation itself is executable. This condition severely limits the compression techniques that can be applied to compact code, and consequently results in poorer compression ratios than unconstrained compression schemes can achieve. Furthermore, when compressing a series of instructions, certain information needs to be retrieved at will. For example, branching and function entry points must be able to be decompressed on demand. This problem has led to the efforts aimed at designing processors with execution modes in shorter instruction formats (e.g., ARM Thumb and MIPS16).

Compression on data may be subjected to different constraints, such that a lossy compression scheme sometimes is acceptable. For example, lossy compressions[41,42] for recompressing the reconstructed reference frames are acceptable for maintaining the random access capability of motion-compensated video coding scheme, such as MPEG-4 and H.264/AVC. Instead of using recompression schemes, a simpler way to reduce the size of the frame buffer is to store scaled pictures[43,44]. Without the memory reduction schemes for the reconstructed reference frames, about 8.9 Mbytes of frame memory could be required for three video frames, each has a frame size of 1920×1080 and 4:2:0 sampling format. As for H.264/AVC, the size of the frame buffer is even more demanding, due to the more reference frames, the more bits per sample, and the larger frame size could be encoded. Therefore, memory reduction schemes for frame buffer will be a more interesting topic for this advanced video coding standard. The tradeoff among picture quality degradation, random access capability, hardware complexity for the reduction schemes, and the latency of reference data extraction should be carefully investigated.

2.4 Dynamic Power Management

Dynamic power management can also be applied to the memory system[45]. In addition to the traditional clock gating approach, Farrahi[46] et al. propose a memory segmentation (also called partitioning) scheme that reduces power by exposing idleness in memory access. Whenever a memory segment is idle, which is the duration when no useful information is stored into it, the memory segment can be put in the sleep mode. Therefore, its clock can be stopped, or its refresh signal can be shut down, thereby minimizing its power dissipation.

Farrahi et al. also propose a worst-case exponential time algorithm for solving the optimization problem of finding the optimum assignments of variables to memory segments that maximize the total idle time of all segments. In their original proposal, instantiating memory segments increases the number of memory components, the area and the wiring overhead. However, this kind of memory segmentation scheme can be considerable without worrying about the aforementioned drawbacks when a DRAM with partial-array self-refresh[47] (PASR) capability is available. PASR capability is already widely supported in mobile RAMs and will be adopted in DDR3 DRAMs.

2.5 Constraint-Aware Target for Memory Systems

We define the constraint-aware target as one in which meeting the constraint goal is a significant design consideration and in which the target modifies its behavior based on current constraint information. As for a traditional target, it is shielded from the detailed information of constraint or has no knowledge of the constraint. Therefore, a constraint-aware target aggressively meets the constraint, whereas the constraint-unaware targets have little improvement or even violate the optimization goals.

For example, Grun et al. present a memory-aware compiler approach that exploits efficient memory access modes by extracting accurate timing information, allowing the compiler's scheduler to perform global code reordering to better hide the latency of memory operations. Their compiler generates aggressive schedules that are on the average 24% smaller than one that assumes no knowledge of memory timing. Marchal et al. propose SDRAM-energy-aware memory allocation for multimedia applications such that the tasks' data is assigned to the available SDRAM banks, thereby reducing the number of page-misses and thus the energy consumption. Lee et al. present a quality-aware memory controller that can meet quality-of-service (QoS) guarantees, which are defined as providing different memory access requirements for fair distribution of bandwidth and shortest possible transaction latency. This memory controller will be introduced in detail in the next section.

3. MEMORY CONTROLLER

As aforementioned, memory access is becoming an important limiting factor with respect to the system performance. It becomes even more critical when data transfer is to off-chip memory. In fact, off-chip memory bandwidth is regarded as the most 'scarce' or 'expensive' resource in Nexperia-DVP products[50]. Unlike the on-chip communication, there are more restrictions on the available 'channels' and protocols for the off-chip memory access. For on-chip communication, the number of physical channels is much more easily increased by using more wires. The protocols for on-chip communication are also more flexible, efficient, and free (i.e., independent). For example, split-transaction or out-of-order returns can be supported whenever necessary. In contrast, the protocols of off-chip memory are administrated by the memory venders and organizations. In addition, the performance of off-chip memory is also significantly affected by the internal architecture of the off-chip memory. It becomes especially critical when DRAMs are used as off-chip memories.

In this section, we first identify the requirement of memory sub-system and the limitations of conventional designs. Then, we introduce the basics of off-chip SDRAM to understand the characteristics of SDRAM access. Finally we present the architecture of our multi-layered quality-aware memory controller. To evaluate the design, we show some experiment results, including some constrained random experiments under different parameters and a simplified STB SoC to examine the QoS performance of several DRAM controllers.

3.1 Memory Sub-System Requirement

Multimedia processing technologies have been widely applied in many systems. These technologies have not only provided existing applications like desktop video/audio but also spawned brand new industries and services like digital video recording, video-on-demand services, high-definition TV, etc. The confluence of hardware and software technologies has given computers the ability to process dynamic media (video, animation, music, etc) where before they could handle only static media data (text, images, and so on). To support complex multimedia applications, architectures of multimedia systems must provide high computing power and high data bandwidth. Furthermore, a multimedia operation system should support real-time scheduling and fast interrupt processing[51].

The tremendous progress in VLSI technology provides an ever-increasing number of transistors and routing resource on a single chip, and hence allows integrating heterogeneous control and computing functions to realize SoCs, the improvement of off-chip communication is limited due to the number of available I/O pins and the physical design issues of these pins. As many recent studies have shown, the off-chip memory system is one of the primary performance bottlenecks in current systems. For example, Hennessy and Patterson show that while microprocessor performance improved 35% per year until 1986, and 55% per year since 1987, the access time to DRAM has been improving about 5% per year[52]. Rattner illustrates that whereas a Pentium Pro requires 70 instruction cycles for a DRAM access, a Pentium 4 running at 2 GHz takes 500 to 600 cycles[53]. Even the performance of DRAM is ever-improved, the system overheads like turnaround time and request queuing account for a significant portion of inefficiencies in memory access. Schumann reports that 30–60% of primary memory latency is attributable to system overhead rather than to latency of DRAM components in Alpha workstations[54]. For multi-core SoC designs, the performance of memory subsystem is even more important, due to the share of memory bus with different access requirements of these heterogeneous cores.

Recognizing the importance of high performance off-chip DRAM communication as a key to a successful system design, several SDRAM controllers and schedulers have been proposed to make the most efficient use of the off-chip DRAM memory subsystem. For single-processor environments, several approaches have been presented to improve memory bandwidth utilization. McKee's Stream Memory Controller (SMC) reorders memory references among streams[55], whereas Rixner's memory bank controller for each DRAM chip reorders both memory references among streams and within a single stream[56]. Several problems come with reordering the DRAM accesses within a single stream. First, the reordering efficiency is quite sensitive to the number of accesses visible to the access scheduler during each clock cycle. Hence, a large amount of register files are needed to hold the arriving DRAM accesses, which inevitably increase the area of the design. Second, since the DRAM accesses may be completed out of order, extra circuits are required to reorder the read data back to the original order to maintain data consistency, which in turn might lead to a reduction in the efficiency of the reorder scheme on the memory bandwidth improvement. Furthermore, these extra circuits also increase the area overhead. Addressing to the aforementioned problem of out-of-order return of read data, Kazushige Ayukawa's[21] access-sequence control scheme allocates an access ID to each DRAM access. Therefore, the processor can identify the original order of the read data according to the access ID. This solution, however, is only suitable for specific bus protocols and processing units (PUs) that have the capability to assign and identify access IDs.

Instead to reorder memory accesses, Tetsuro Takizawa presented a memory arbiter to increase the bandwidth utilization by reducing bank conflicts (i.e., row miss) and read/write turnarounds for the multi-core environment[57]. The arbiter lowers the priority of a DRAM access if the access is addressed to the same bank as the previous granted access, or the access direction (read or write) is different from that of the previous granted one. Hence, the possibility of bank conflicts and read/write turnarounds can be diminished.

For the multi-processor vector machine with multi-port memory system, Corbal proposed Command Vector Memory System (CVMS) to reduce the processor to memory address bandwidth by sending commands to the memory controllers as opposed to sending individual addresses[58]. In CVMS, a command, including a base address and a stride, is expanded into the appropriate sequence of references by each off-chip memory bank controller.

The above-mentioned SDRAM controller designs only address to the improvement of the overall bandwidth utilization. However, PUs in a heterogeneous system usually require different services of memory access bandwidth and latency. Therefore, these SDRAM controllers need to cooperate with DRAM schedulers to provide proper DRAM services for these PUs. In our observation, traditional DRAM scheduler designs have one or more of the following limitations:

- the unawareness of DRAM status leading to low scheduling efficiency on both DRAM bandwidth utilization and access latency,
- the lack of control over the bandwidth allocation for different PUs (e.g., fixed-priority scheduler) leading to starvation of low-priority PUs in some situations, and
- the significant access latencies due to the fair scheduling policies (e.g., round-robin scheduler) leading to unbearable long access latencies for high-priority PUs.

Knowing the above limitations of the conventional DRAM scheduler designs, Sonics' MemMax memory scheduler provides quality-of-service guarantees of a single, shared off-chip DRAM memory subsystem for multiple heterogeneous functional units by using a tiered filtering system[59]. However, MemMax is deeply coupled with Sonics' SiliconBackplane μNetwork that is time-shared with these functional units. Access ownership of SiliconBackplane μNetwork is determined by a two-level arbitration scheme: the first level of arbitration is based on a Time-Division Multiplexing (TDM) mechanism, while a second, lower priority access scheme is based on a round-robin token passing mechanism[60]. Due to the inherent latency limitations of the TDM on-chip communication mechanism[61], MemMax cannot effectively provide short latency services and only has better improvement on bandwidth utilization. Similarly, a three-level memory arbitration scheme proposed by Harmsze[62] can be used for systems where both continuous high-throughput and random low-latency requests present. However, this three-level memory arbitration scheme does not take the status of SDRAM into account. In addition, the first-come-first-serve scheme used for the arbitration of the continuous streams also lowers the memory bandwidth utilization.

Instead of using a single-port architecture to connect to system bus, Denali's Databahn has a multi-port architecture to ensure memory is shared efficiently among many high-bandwidth client modules[63]. Databahn has three serial-connected engines to maximize memory access performance:

- a priority engine to prioritize and weight port traffics by selecting among bandwidth-weighted algorithms,
- an ordering engine to reorder commands to optimize bandwidth while maintaining relative priority and memory coherency, and
- a sequencing engine to sequence data commands to minimize lost transaction cycles and latency.

However, since the architectures are proprietary, underlying algorithms are unknown.

The aforementioned critical issues in integrating heterogeneous control and computing functions into a single chip motivate us to explore an efficient solution of off-chip SDRAM memory controller for multimedia platform SoCs[64]. The goal of this memory controller is to provide not only the high utilization of DRAM bandwidth bus also the quality-of-service (QoS) guarantees, which are defined as:

- fair distribution of bandwidth over bandwidth-sensitive PUs
- shortest possible transaction latency for latency-sensitive PUs.

In addition, most multimedia PUs have regular address patterns. Having built-in address generators in the memory controller can reduce the address bus traffic and therefore increase the efficiency of on-chip communication. On the other hand, because not every system needs the same requirement of memory usage, a well-partitioned architecture of a memory controller can let system designers choose and integrate the required functionality of the memory controller into their systems. Based on these requirements, a multi-layered, quality-aware memory controller with the following features will be presented in this section.

- A layered architecture of the memory controller that can efficiently decouple different functionality, including memory-specific control, quality-aware scheduling, and built-in address generators into different layers of the memory controller. Different combinations of these layers can produce memory controllers with different capabilities to meet distinct system requirements.
- A high efficient and flexible SDRAM memory interface socket (MIS) to take charge of SDRAM-specific control and make the best use of SDRAM bandwidth by supporting parallel access of each bank within SDRAM. In addition, MIS can respond to the requests from the SDRAM scheduler immediately to furthermore make the best use of SDRAM command and data bus. On the other hand, the flexibility of MIS is based

on the configurable, shared-state FSM design that can easily be adjusted for different complex timing control latencies of SDRAM.

- A quality-aware scheduler (QAS) that not only improves the SDRAM bandwidth utilization by considering the SDRAM status and the relations of SDRAM accesses, but also provides QoS guarantees, i.e., minimum access latency and guaranteed bandwidth services based on the memory access requirements of different PUs.

3.2 SDRAM Basics

Figure 1 shows a simplified architecture of a two-bank SDRAM. All memory banks share the data and address bus, whereas each bank has its own row decoder, column decoder and row buffer. The mode register stores several SDRAM operation modes such as burst length, burst type, CAS (column address strobe) latency, etc. An *m*-bank SDRAM has a similar architecture. A complete SDRAM access may consist of several commands including row-activation, column-access (read/write) and precharge, as shown in Figure 2. A row-activation command, together with the row and bank address, is used to open (or called activate) a specific row in a particular bank, and copy all data in the selected row into the selected bank's row buffer for the subsequent column accesses. After accepting this command, SDRAM needs a latency called t_{RCD} (ACTIVE to column access delay) to accomplish the command. No other commands can be issued to this bank during this latency. However, commands to other banks are permissible

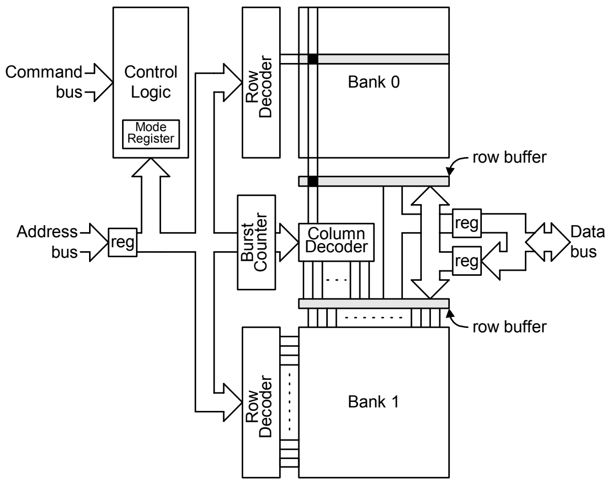


Figure 1. A simplified architecture of a two-bank SDRAM

due to the independent parallel processing capability of each bank. Once a row of a particular bank has been opened, a column-access command can be issued to read/write data from/to the addressed word(s) within the row buffer. To issue either a read or write column-access command, DRAM address bus is also required to indicate the column address of the open row in that bank. For a write access, DRAM data bus is needed to transfer write data from the time the command is issued until the whole burst transfer is completed. As for a read access, DRAM data bus is used to transfer data after a latency called CAS, which is the time from the read column-access command is registered to the first read datum is available. The precharge command, together with the information on address bus, can be used to deactivate a single open row in a particular bank or all rows in all banks. While processing the precharge command, the addressed bank or banks are not allowed to accept any other commands during a time called t_{RP} (PRECHARGE command period.)

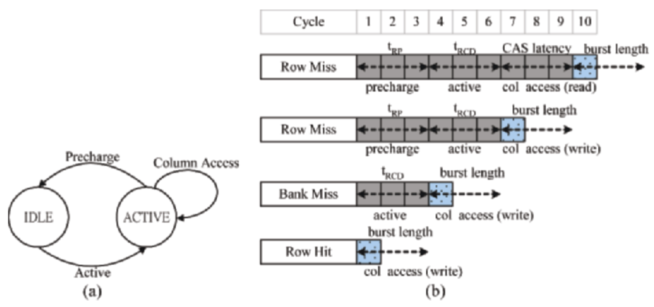

Precharge
Column Access
IDLE
ACTIVE
Active
(a)
Cycle 1 2 3 4 5 6 7 8 9 10
Row Miss
t_{RP} t_{RCD} CAS latency burst length
precharge active col access (read)
Row Miss
t_{RP} t_{RCD} burst length
precharge active col access (write)
Bank Miss
t_{RCD} burst length
active col access (write)
Row Hit
burst length
col access (write)
(b)

Figure 2. (a) Simplified bank state diagram, and (b) access latencies of different access statuses

SDRAM bandwidth utilization and latency is lower and longer respectively when more commands are required for a SDRAM access. The number of commands needed for a complete SDRAM access deeply depends on the state of the bank addressed by the SDRAM access. Figure 2(a) and Figure 2(b) show a simplified bank state diagram and the access latencies due to different access statuses: bank miss, row miss, and row hit. In a bank miss status, an incoming access is addressed to a bank in the IDLE state, therefore it must first activate the target row and then issue the column access command. For a row miss status, the addressed bank is in ACTIVE state and the row address of its activated row is not identical to that of an incoming access. In this case, the incoming access has to first precharge the bank, then activate the target row, and finally issue column-access commands. As for a row hit status, the addressed bank is in ACTIVE state and the row address of its activated row is the same as that of the incoming access. Hence, column-access commands can be directly issued. The above discussion is only based on a simplified condition. A more complete discussion on various access latencies of a SDRAM access can be found in Lee's paper[65].

3.3 Multi-Layered, Quality-Aware Memory Controller

Figure 3 shows the configurations of different layers of the proposed memory controller. Layer 0 memory interface socket (MIS) is a configurable, programmable, and high-efficient SDRAM-specific controller for basic SDRAM operations, such as SDRAM initialization, refresh function, etc. Basically, MIS accepts access requests and translates them into proper command sequences according to the DRAM access status mentioned in the preceding section. We[65] have designed a self-generating, tool-independent MIS silicon intellectual property (IP) to alleviate the burden for system designers. This IP, called MIS-I in this paper, is featured with its parameterized and blockwised design which is characterized by a rich set of choices of functionality, performance, interface and testbench. To improve SDRAM bandwidth utilization and access latency, an improved MIS-I, called MIS-II, is presented in this paper. Together with Layer 1 quality-aware scheduler (QAS), the memory controller also has the capability to provide QoS guarantees for heterogeneous control and computing functional units in multimedia SoC designs. Moreover, Layer 2 built-in address generator (BAG) designed for multimedia PUs can effectively reduce the address bus traffic and therefore further increase the efficiency of on-chip communication.

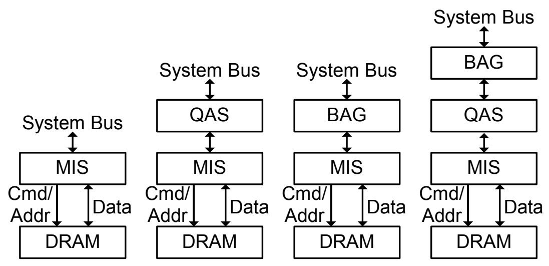


Figure 3. Configurations of different layers of the proposed memory controller

3.3.1 Configurable, shared-state FSM design

Traditionally each control state of a FSM occupies one state of the FSM. However, this design style makes the FSM less flexible when there are many repeated control states in a FSM. For example, to complete a SDRAM read access with burst length of four in the row miss status, the timing diagram and a conventional FSM design for this access are shown in Figure 2(b) and Figure 4(a) respectively. These control latencies depend on both the specification provided by the SDRAM vendor and the clock frequency the memory is clocked. In addition, the unit of these latencies specified in the SDRAM datasheet is on the basis of real time (e.g., nanoseconds and microseconds), whereas the design of FSM that controls this sequence is clock cycle based. When the system clock is varied, these control latencies are varied in terms of clock cycle count. Conventional FSM design manually calculates the control latencies from the relation between the timing constraints in the datasheet and the clock frequency of the target system, and then fixes these latencies as states shown in Figure 4(a). Changes in the control latencies or the numbers of access data force us to manually modify the design of FSM. For a flexible and reusable design, these hard-coded states should be reduced or eliminated.

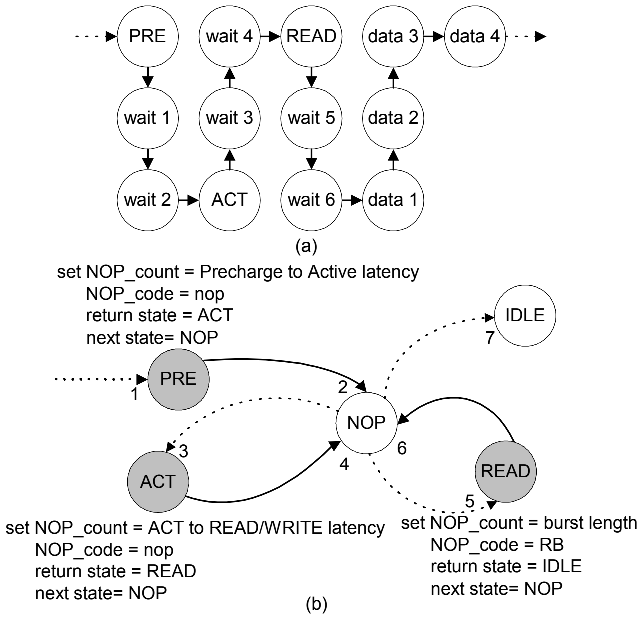


Figure 4. FSM design in (a) traditional DRAM controller, and (b) MIS-I

To make FSM more flexible, MIS unifies the repeated control states into a single control state of FSM as indicated in Figure 4(b). Numerous ‘wait’ states are needed to handle DRAM command latencies. In MIS-I, these states are all mapped to one ‘NOP’ (No OPeration) state. Before entering the NOP state, several registers have to be set by the command states (gray ones). These registers are *NOP_count* (the cycle count which is needed to stay in the NOP state), *NOP_code* (operation mode of MIS while in NOP state), and *return state*. In addition to wait states, data transfer states can also be mapped to NOP state and NOP_count now becomes the programmed DRAM burst length. Since MIS combines all wait and data transfer states into a NOP state and loads the command latencies or burst length into NOP_count dynamically, it is very easy to parameterize the command latencies without redesigning the FSM. If control latencies are determined and fixed before synthesis, they become hardwired logic after synthesis. These control latencies and other related SDRAM timing constraints are automatically converted from absolute timing to cycle count through built-in mathematical equation in a Verilog script file. In contrast, if there is a requirement of change in memory or clock frequency after the whole system is designed, control and status registers are allocated for these latencies to enable the ability to program them dynamically.

3.3.2 MIS-II

In MIS-I, it is clear that when the first DRAM access request is handled by the single, shared FSM, the second one has to wait until the first access has been completed. This procedure works fine for successive accesses addressed to the same bank since DRAM can't process them at the same time. However, for those accesses addressed to different banks, memory bandwidth loss is unavoidable. Because all repeated states of MIS-I are merged into a single shared NOP state, the performance of MIS-I is constrained by its poor capability of parallel processing accesses addressed to different banks. Figure 5(a) shows how MIS-I processes two row-miss accesses addressed to different banks. It is obvious that before the command latency of a registered command has been met, no other commands can be issued. Hence, long access latencies and low bandwidth utilization are

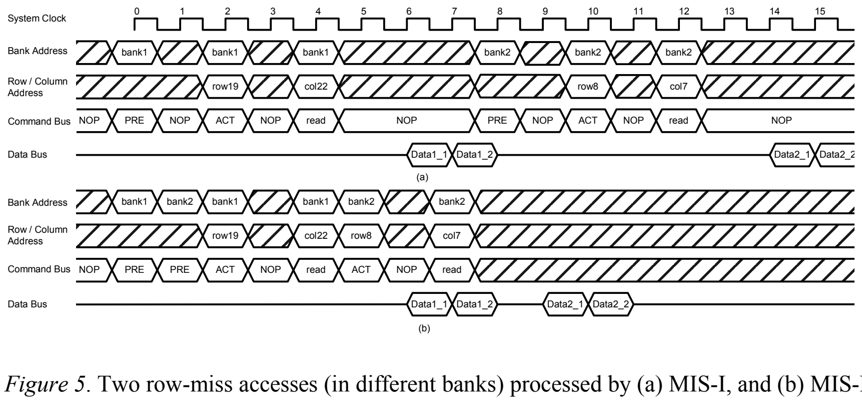

System Clock
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Bank Address
bank1 bank1 bank1 bank2 bank2 bank2
Row / Column Address
row19 col22 row8 col7
Command Bus
NOP PRE NOP ACT NOP read NOP PRE NOP ACT NOP read NOP
Data Bus
Data1_1 Data1_2 Data2_1 Data2_2
(a)
Bank Address
bank1 bank2 bank1 bank1 bank2 bank2
Row / Column Address
row19 col22 row8 col7
Command Bus
NOP PRE PRE ACT NOP read ACT NOP read
Data Bus
Data1_1 Data1_2 Data2_1 Data2_2
(b)

Figure 5. Two row-miss accesses (in different banks) processed by (a) MIS-I, and (b) MIS-II

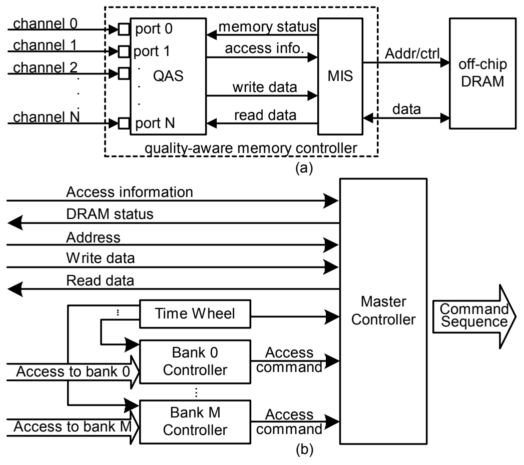


Figure 6. (a) Quality-aware memory controller architecture, and (b) MIS-II architecture

expectable. By using bank controllers and the master controller, MIS-II effectively fixes these problems (see Figure 5(b)) and still inherits the advantages of MIS-I. As indicated in Figure 6(b), each internal bank of SDRAM is allocated one bank controller to process accesses addressed to that bank. After accepting an access from the input port, the bank controller generates appropriate SDRAM commands according to the timing information provided by the time wheel. SDRAM commands from all bank controllers are collected by the master controller, which then issues the most proper command to SDRAM according to the information from time wheel and QAS. Master controller is also responsible for all of the other control procedures such as power-up, refresh, etc. To regulate the processing and sequencing of all control within and among bank controllers and the master controller, the time wheel has to be carefully designed. Roughly speaking, MIS-I can be treated as MIS-II with only one bank controller.

To enhance the parallel processing of QAS, the access channels are connected as a star topology, which is widely used in practical SoC platforms, such as ARM's PrimeXsys platforms[66] and Palmchip's CoreFrame architecture[67]. Each channel has a dedicated bus connected to each port of QAS, while several PUs may share a channel. The share of a channel is basically based on the memory access characteristics of PUs, which will be illustrated in detail later.

Another mechanism that makes MIS-II more efficient than MIS-I is to separate burst transfer control from bank controllers. As mentioned earlier, data transfer states of MIS-I is merged into the shared NOP state. After all burst data have been transferred, the FSM returns to IDLE state and readies for accepting the next access. If the next access has already been pending, it still has to wait at least one clock cycle on state transition after the current access is completed. In addition, preparation cycles are sometimes needed for DRAM controllers to preprocess an access. These two types of delay result in unwanted bandwidth loss. In MIS-II design, the burst transfer control is handled by the master controller. After issuing the column-access command, the bank controller can return to IDLE state and accept the next access. Master controller will generate signals needed during the burst transfer cycles. Hence, the state transition and preparation cycles can be overlapped with the burst transfer cycles and the bandwidth loss is mitigated. In brief, MIS-II is designed to respond to the DRAM access requests immediately to furthermore make the best use of SDRAM command and data bus. Therefore, SDRAM bandwidth utilization is raised whereas access latency is diminished.

3.3.3 Built-in address generator

The efficiency of on-chip communication plays an important role on system performance. For PUs having regular access behaviors, (e.g., image processing unit, audio/video codec, etc.) the addresses of their DRAM accesses can be obtained in advance or through a simple translation. BAG is designed to generate access addresses locally for some multimedia processing units. BAG can be connected either to QAS or MIS. Without transferring address information for every DRAM access, the address bus traffic of on-chip channel can be effectively reduced. Currently, the following address generators are supported in our design.

- 1-D (linear) address generator: By giving the start address and access length, the 1-D address generator can automatically generate addresses for PUs whose address mapping schemes are linear, e.g., audio codec, according to the programmed DRAM burst length.
- 2-D block based address generator: The 2-D block based address generator is designed for block based processing units such as MPEG-2 motion compensation, DCT, etc. Tile-based mapping of behavioral array references to physical memory locations is used to minimize power consumption on address bus transitions[28] and improve DRAM utilization[57].

3.4 Quality-Aware Scheduling

Access conflicts of shared resources are an old problem in hardware design. Mechanisms such as semaphores and scheduling are conveniently applied to eliminate these conflicts. Two common used scheduling policies are round-robin and fixed-priority scheme. Round-robin policy can fairly allocate

DRAM bandwidth to all channels. However, lacking priority nature makes it hard to guarantee access latency for any channel. Fixed priority policy may solve the problem of access latency. However, the lower-priority channels may suffer starvation due to high access rates of higher priority channels. Thus, it is obvious that neither of these two scheduling policies can effectively provide QoS guarantees. This problem can be especially fatal to some applications such as multimedia system designs. For example, signal processing units such as video codec may require guaranteed bandwidth, whereas CPU may concern about the access latency more when waiting for a cache line fetch. To effectively solve this problem, we propose a quality-aware scheduler whose scheduling policy can provide not only high DRAM utilization but also QoS guarantees. In the proposed quality-aware scheduler design, channels are put into three categories: latency-sensitive, bandwidth-sensitive, and don't care.

1) Latency-Sensitive Channel: Latency-sensitive channels are for PUs that are highly concerned about latencies of DRAM accesses. Accesses issued through latency-sensitive channels are called latency-sensitive accesses. Normally latency-sensitive accesses will be granted with the highest priority. Even though in this case, access latencies may still be long due to DRAM's status. For example, if a latency-sensitive access is addressed to a DRAM bank that is currently busy in serving another access, it won't be granted until the bank returns to standby status when using DRAM controllers with conventional schedulers, such as MemMax DRAM scheduler[59]. Even when the addressed bank is at standby status, the command or the data bus of DRAM may be occupied by other accesses having been granted to access to other banks. The situation is more severe when DRAM is set at long burst mode. In order to reduce those latencies caused by the aforementioned conditions, QAS provides two services to shorten the DRAM access latencies of latency-sensitive accesses: preemptive and column-access-inhibition (CAI) services.

- Preemptive service
  Preemptive service is used to issue latency-sensitive accesses as soon as possible by suspending the processed access from a bandwidth-sensitive or don't care channel. This indicates that preemptive service may reduce the average bandwidth utilization. The problem is worse when the previous access has already activated a distinct row in the same bank. Naturally, latency-sensitive accesses that are being processed will be protected and won't be suspended by other latency-sensitive ones.

- Column-access-inhibition service
  CAI service is used to preserve the data bus for latency-sensitive accesses by inhibiting issuing column-access commands from bandwidth-sensitive and don't care channels, and therefore eliminate latencies resulted from data bus congestion. Again, the overall bandwidth utilization is

diminished when CAI service is applied since the data bus is not optimally utilized.

Besides the above two services, the bank controller which is processing latency-sensitive accesses also has the highest priority to use DRAM command bus to avoid possible latencies caused by waiting for the command bus. Because optimizing the access latencies for some particular channels is harmful to the overall bandwidth utilization, preemptive and CAI service for latency-sensitive channels are only guaranteed within an allocated DRAM bandwidth. Access requests beyond this allocated bandwidth will change the channel to don't-care type. The trade-off between high bandwidth utilization and short access latencies for some particular channels should be thoroughly considered by the system designers.

2) Bandwidth-Sensitive Channel: Bandwidth-sensitive channels are for PUs that concern only about bandwidth. Since accesses through these types of channels are insensitive to latency, they are scheduled by DRAM status (i.e., access status and Read/Write turnaround) to achieve the highest bandwidth utilization. For example, a row-hit access will have higher priority than a row-miss access. If two accesses have the same bank status, QAS will grant them according to round-robin scheduling policy that can fairly favor all accesses in different channels. QAS also gives those accesses having the same access type (read or write) as the previously granted one the higher priority to prevent bandwidth loss due to SDRAM data bus turnaround cycles. The access requests within the allocated bandwidth will be fulfilled by QAS to provide guaranteed bandwidth for each channel. Access requests beyond the allocated bandwidth, however, will change the channel to don't-care type.

3) Don't-Care Channel: Don't-care channels are for PUs that care about neither latency nor bandwidth. Accesses through these types of channels won't be guaranteed any bandwidth or latency. They are processed only when extra bandwidth is left over by the first two types of channels. The same as bandwidth-sensitive channels, accesses through don't-care channels are scheduled by DRAM status and may be suspended by preemptive service.

The bandwidth allocation for latency-sensitive and bandwidth-sensitive channels is on the basis of service cycle. Service cycles are cycles in which data are transferred to/from DRAM. As shown in Figure 7, a service period

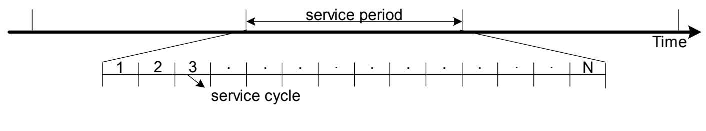


Figure 7. Definition of service cycle and service period

is a union of N service cycles. System integrators can configure the number of service cycles for different channel types in one service period by using the scheme originally proposed for bus management[68]. By assigning service cycles for latency- and bandwidth-sensitive channels, users can have fine-grained control over the bandwidth allocated to these types of PUs.

Figure 8 shows the pseudo-code of the proposed quality-aware scheduling. At the beginning of a service period, all channel services are enabled. These services won't be disabled until running out of the allocated bandwidths (service cycles actually) in a service period. At each cycle, QAS checks requests from all channels *C*. Requests from latency-sensitive channels $C_{\text{LS\_list}}$ are served first. When there are multiple requests from $C_{\text{LS\_list}}$, they are scheduled by round-robin scheduling policy to decide the final unique request C_{winner}. If there is already a latency-sensitive access which is being served, C_{winner} stays in pending status. Otherwise, C_{winner} is served with preemptive and CAI services. When there is no request from $C_{\text{LS\_list}}$, requests from bandwidth-sensitive channels $C_{\text{BS\_list}}$ are served. When

```
check_channel_request($C$);

if (LS_channel_assert) { //latency sensitive channel
  for ($C_i$ in $C_{\text{LS\_list}}$)
    if (check_allocated_BW($C_i$) ==NULL)
      change_channel_type(donot_care, $C_i$);
  $C_{\text{winner}}$=round_robin($C_{\text{LS\_list}}$);
  if (preemptive_service_running or CAI_service_running)
    MIS_service(pending, $C_{\text{winner}}$);
  else
    MIS_service(preemptive, CAI, $C_{\text{winner}}$);
}
else if (BS_channel_assert) { //bandwidth-sensitive channel
  for ($C_i$ in $C_{\text{BS\_list}}$)
    if (check_allocated_BW($C_i$) ==NULL)
      change_channel_type(donot_care, $C_i$);
  sorted_by_DRAM_status($C_{\text{BS\_list}}$);
  $C_{\text{winner}}$=round_robin($C_{\text{BS\_list}}$);
  MIS_service(normal, $C_{\text{winner}}$);
}
else if (donot_care_channel_assert) {
  sorted_by_DRAM_status($C_{\text{donot\_care\_list}}$);
  $C_{\text{winner}}$=round_robin($C_{\text{donot\_care\_list}}$);
  MIS_service(donot_care, $C_{\text{winner}}$);
}

if (end_of_a_service_period)
  reset_all_channel_setting();
```

Figure 8. Quality-aware scheduling

there are multiple requests from $C_{\text{BS\_list}}$, they are scheduled by DRAM status. Round-robin scheduling policy is then applied to those requests with the same DRAM status to decide the final unique request C_{winner}. Finally, requests from don't-care channels are accomplished only when no requests from $C_{\text{LS\_list}}$ and $C_{\text{BS\_list}}$, and the final unique request C_{winner} of these requests is decided in a similar manner for $C_{\text{BS\_list}}$.

3.5 Experimental Result

In this section, we present the experimental framework used to evaluate several SDRAM controllers. We will describe a system test-bed and the use of each component in this test-bed. First, several constrained random experiments are conducted under different configuration parameters to measure the average SDRAM bandwidth utilization and the access latencies for the highest priority access channels or latency-sensitive channels of the considered SDRAM controllers. Then, a simplified STB SoC is simulated to examine the QoS performance of each SDRAM controller.

3.5.1 Experimental framework setup

Figure 9 shows the test-bed used to evaluate the considered SDRAM controllers. Each access initiator generates different SDRAM access behaviors for each channel connected to the SDRAM controller according to three control parameters: *process_period* indicates how many clock cycles an access initiator needs to process the read data; *access_num* indicates how many accesses an initiator issues every *process_period*; *access_behaviors* specifies different access patterns: constrained random, 1-D linear, 2D block base, 2-D interlace, and 2-D block base with unpredictable start address. The simulation coordinator is responsible for generating all control signals needed in the experiments and dumping the simulation results for further analysis. Two types of on-chip bus (OCB) model are used in the

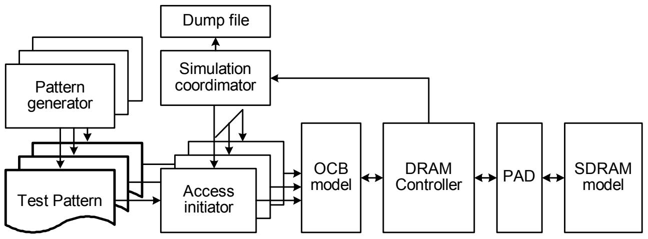


Figure 9. Test-bed for SDRAM controller performance evaluation

experiments: a single shared bus and multiple dedicated buses. The single shared bus model is used to connect a single-channel SDRAM controller and the access initiators. In this model, a bus arbiter is included to grant accesses from access initiators with one clock cycle bus handover. The applied arbitration policies are round-robin and fixed-priority. Multiple dedicated buses are used to connect a multi-channel SDRAM controller and the access initiators. Seven different SDRAM controllers listed in Table 1 will be compared in the following experiments. The SDRAM model used in this experiment is Micron's mt48lc8m16a2 SDR-SDRAM[69]. Some key parameters of this SDRAM are listed in Table 2.

Table 2. SDRAM controllers used in the experiments

Controller	Descriptions
SIG-RR-MIS-I	Single-channel MIS-I controller with round-robin OCB arbiter
SIG-FP-MIS-I	Single-channel MIS-I controller with fixed-priority OCB arbiter
SIG-RR-MIS-II	Single-channel MIS-II controller with round-robin OCB arbiter
SIG-FP-MIS-II	Single-channel MIS-II controller with fixed-priority OCB arbiter
MUL-RR-MIS-II	Multi-channel MIS-II controller with round-robin DRAM scheduler
MUL-FP-MIS-II	Multi-channel MIS-II controller with fixed-priority DRAM scheduler
QA-MIS-II	Multi-channel MIS-II controller with quality-aware DRAM scheduler

Table 3. Key parameters of SDRAM model

Parameters	Values	Parameters	Values
Clock rate	100 MHz	t_{RCD}	2 cycles
Data bus width	16 bit	t_{RP}	2 cycles
Num. of bank	4	t_{RAS}	5 cycles
Burst length	1, 2, 4, 8	CAS latency	2 cycles

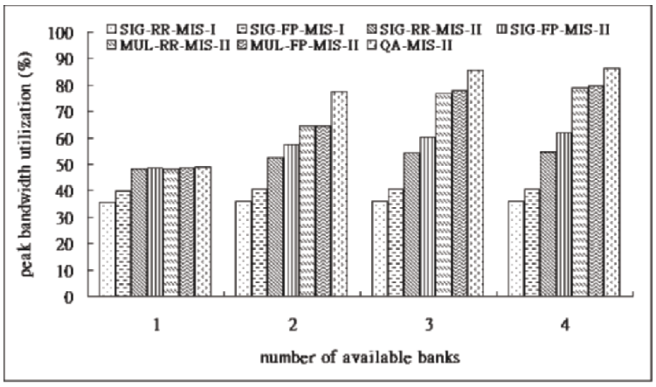
3.5.2 Performance evaluation of constrained random access streams

Performance evaluation of constrained random access streams looks at the performance variations of SDRAM controllers when some control parameters are changed. The default control parameters are listed in Table 3, whereas some of them may vary in different experiments. In every experiment, the bandwidth utilization provided by each SDRAM controller will be observed. Besides, to measure the shortest access latency (denoted as *min_latency*) provided by each SDRAM controller, we assume that access from *initiator 0* will be granted with the highest priority in fixed-priority controllers and set as the latency-sensitive channel in the quality-aware controllers. For round-robin controllers, no particular settings are needed due to the fair scheduling.

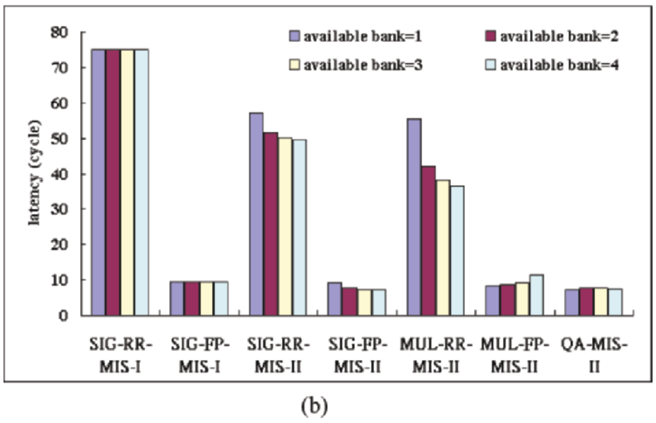
1) Effect of Available Bank

For the SDRAM containing multiple banks, the capability of SDRAM controllers to handle parallel bank access can severely affect the DRAM bandwidth utilization. Figure 10(a) illustrates the bandwidth provided by each SDRAM controller when the number of available banks varies from one to four. Some key points are listed as followed.

- For single-channel MIS-I controllers, increasing the DRAM banks obviously contributes nothing to bandwidth utilization. It is mainly because that MIS-I does not support parallel bank access. Furthermore, FP-MIS-I controller has better performance than RR-MIS-I controller has. This is evident since fixed-priority policy allows high-priority access initiators to occupy the MIS-I longer and hence avoids wasted cycles due to frequent bus handover.
- For one-bank SDRAM, since every SDRAM access cannot be issued until the previous one is completed, bus handover cycles shorter than the burst length have totally no influence on the bandwidth utilization. Hence, all MIS-II controllers provide the same bandwidth. However, the



(a)



(b)

Figure 10. The effects of number of available bank on (a) bandwidth utilization, and (b) *min_latency* hidden bus handover and scheduling cycles can still make MIS-II controllers provide better bandwidth utilization than MIS-I controllers do.

- The bandwidth utilization of MIS-II controllers increases when the number of available banks increases. Compared to single-channel MIS-II controllers, multi-channel controllers have more improvement on bandwidth due to the support of high-efficient parallel bank access.

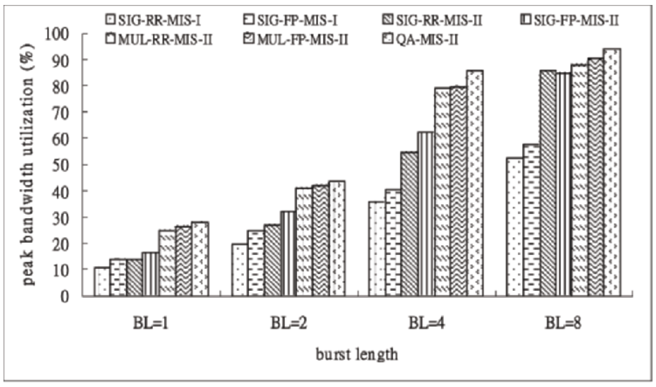
The effect of bank number on *min_latency* of each SDRAM controller is shown in Figure 10(b). We make the following observations from this figure.

- *min_latency* of single-channel MIS-I controllers are insensitive to the number of available bank since no parallel bank access is supported. In contrast, *min_latency* of single-channel MIS-II controllers are reduced when the number of available banks is increased.
- For the MUL-FP-MIS-II controller, increasing the available banks makes the access latencies longer. This is because multi-channel controllers allow accesses addressed to a free bank to be processed as soon as possible to optimally utilize the DRAM bandwidth. Hence, low-priority accesses and high-priority ones may be processed at the same time. This condition unavoidably makes the command and data bus congestion more serious, which in turn leads to longer *min_latency*. This problem is effectively eliminated by the preemptive and CAI services used in QA-MIS-II controller.

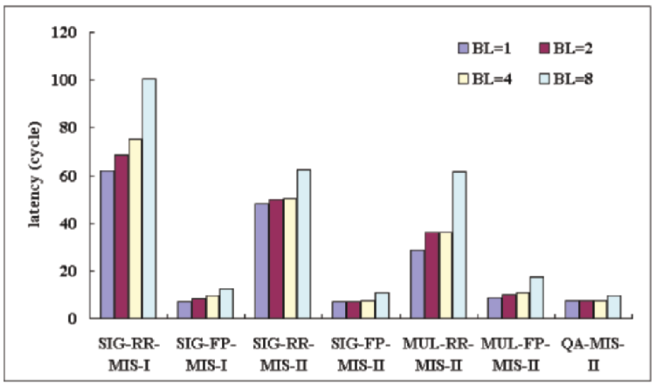
2) Effect of Burst Length

Generally speaking, increasing the burst length comes with higher bandwidth utilization since each column-access command can transfer more data and therefore bandwidth loss due to bus handover and turnaround is reduced. Figure 11(a) depicts the bandwidth provided by different SDRAM controllers when the burst length varies from one to eight. As we can see, single-channel MIS-II controllers can provide almost the same bandwidth utilization as the multi-channel controllers can when the burst length was programmed to eight. It is reasonable since the burst transfer cycle is long enough for single-channel controllers to hide all bus handover cycles and command latencies.

Figure 11(b) illustrates *min_latency* of *initiator 0*. For single-channel controllers, increasing burst length results in longer access latency since the time to wait for the completion of the previous accesses is longer. As for multi-channel controllers, increasing the burst length can be taken as increasing the data bus congestion. Hence, the access latencies is also longer. Compared to MUL-FP-MIS-II, *min_latency* of QA-MIS-II is reduced by 46% and 30% when the burst length is programmed to eight and four respectively. These improvements are due to that SDRAM data bus congestion problem



(a)



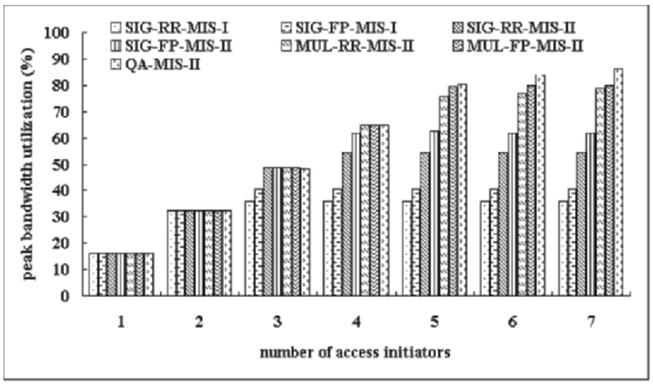
(b)

Figure 11. The effects of burst length on (a) bandwidth utilization, and (b) *min_latency*

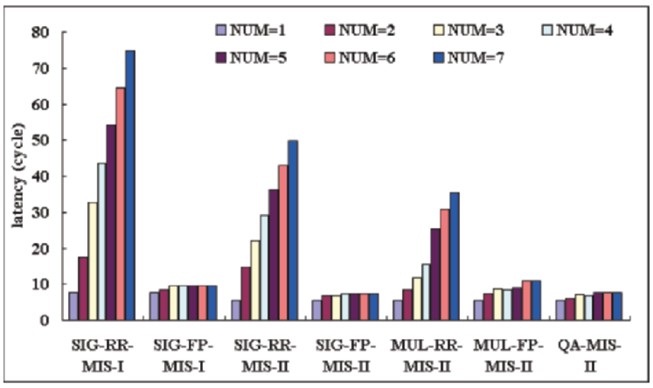
happened to multi-channel controllers and are again eliminated by the preemptive and CAI services.

3) Effect of Number of Access Initiators

Figure 12(a) and Figure 12(b) show the effects of the number of access initiators on bandwidth utilization and *min_latency* respectively. When the number of initiators is less than two, the total bandwidth requirement can be fulfilled by all controllers. The performance difference starts to be obvious when more than three initiators are concurrently issuing DRAM accesses. Single-channel MIS-I and MIS-II controllers encounter their performance bound when the number of initiators is three and four respectively. In contrast, multi-channel controllers still have some available bandwidth for initiators more than four and the bandwidth utilization even goes higher with more access initiators. Figure 12(b) shows *min_latency* of each SDRAM controller. It is clear that when the number of the initiators increases, the access latencies become longer. Compared to MUL-FP-MIS-II, *min_latency*



(a)



(b)

Figure 12. The effects of channel number on (a) bandwidth utilization, and (b) *min_latency*

of QA-MIS-II is reduced up to 30% because the congestion problem can be solved by preemptive and CAI services.

4) Effect of Preemptive and CAI Service

As shown in Figure 13, we take a look at how preemptive and CAI services affect the overall system performance. Compared to MUL-FP-MIS-II, any configuration of QA-MIS-II can provide higher bandwidth utilization due to the high-efficient scheduling for accesses from bandwidth-sensitive channels. When both the preemptive and CAI services are enabled, the bandwidth degradation is most severe compared to other configurations. The second severe degradation of bandwidth occurs when only CAI service is enabled since this service preserves the DRAM data bus for latency-sensitive

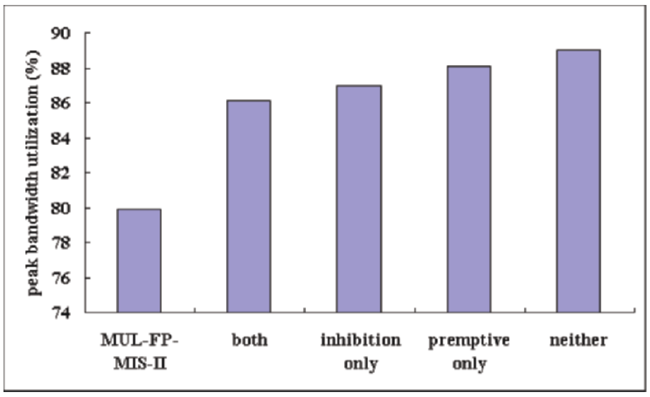


Figure 13. Bandwidth comparison between MUL-FP-MIS-II and QA-MIS-II with different services

channels to prevent data bus congestion problem and therefore results in low bus utilization. When only preemptive service is enabled, the bandwidth degradation is minor. This is because preemptive service can make accesses from *initiator 0* be processed as soon as possible and then free QAS to grant other requests from non-latency-sensitive channels. This higher efficient scheduling is helpful to increase bandwidth utilization and hence can neutralize the bandwidth loss resulted from preemptive service.

Figure 14 illustrates *min_latency* of MUL-FP-MIS-II and QA-MIS-II with different services. It is obvious that optimizing the access latency of a particular channel is harmful to that of other channels. Note when both preemptive and CAI services are disabled, *min_latency* of QA-MUL-II is

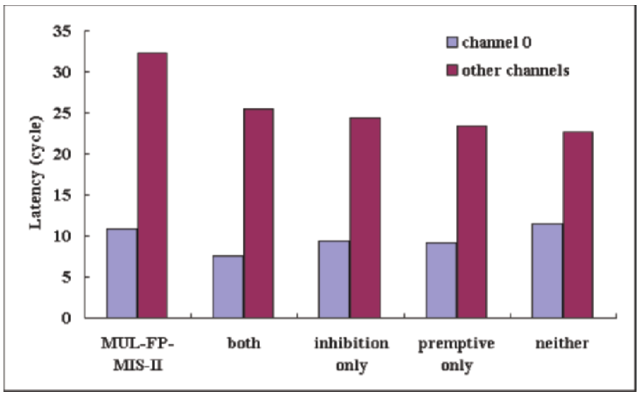


Figure 14. Latency comparison between MUL-FP-MIS-II and QA-MIS-II with different services

slightly longer than that of MUL-FP-MIS-II. This is because the higher bandwidth utilization of QA-MIS-II also makes the data bus congestion more severe. The access latency of other channels of MUL-FP-MIS-II is higher than that of QA-MIS-II because it is dominated by access latencies of low-priority channels.

5) Summary

The above experiments clearly indicate that multi-channel SDRAM controllers can provide much higher bandwidth than single-channel controllers when high-efficient parallel bank access is supported. Furthermore, these experiments also show that maximizing the bandwidth utilization can be harmful to access latency for some particular initiators because of the possible command and data bus congestions. This problem can be effectively solved by preemptive and CAI service provided by QA-MIS-II. Hence, only QA-MIS-II can successfully provide both high bandwidth utilization and short access latency services.

3.5.3 Performance evaluation of set-top-box emulation environment

In this section, we simulate several events that may occur in a digital STB chip, which is a good example of multimedia SoC design. The basic hardware components of a STB SoC may include a CPU, audio/video codec, network devices, etc. Various demands on DRAM service cause it difficult to design a memory controller that fulfills different requirements of these components. Some new features supported by digital STB, such as the interactive television (ITV) service, make the situation even worse due to the drastic change in the requirement of DRAM bandwidth. ITV service allows users to not just sit in front of the TV but interact with the broadcasting programs, such as online betting, home shopping, etc. Therefore, the DRAM bandwidth required by CPU might vary when CPU performs different application programs.

1) System Setup

In this experiment, seven PUs with configuration listed in Table 4 share a unified off-chip SDRAM. To assure low-latency access for CPU’s cache line fetch, CPU is taken as latency-sensitive PU in the quality-aware controller and the highest priority in fixed-priority controllers. Besides, because users are less sensitive to the efficiency of download speed, wireless LAN controller is set as the lowest priority and don’t care PU in fixed-priority and quality-aware controllers respectively. Since round-robin controllers fairly schedule accesses from all PUs, no special configuration is needed. Figure 15 shows some events that may occur when the STB operates. Most of the time the user just watches TV programs. Therefore, the total bandwidth

Table 4. Control parameters in constrained random experiments

Parameters	Values
No. of access initiators	7
No. of bank used	4
Burst length	4
Process_period	60
access_no	3
Bandwidth requested by each initiator	32.4 MB/s

requirement of the STB system is rather steady. While watching the program, the user also downloads some files through wireless LAN, e.g., video clips or MP3 files. Two events that cause variation of SDRAM bandwidth requirement are on-screen display (OSD) and ITV events. The OSD event activated by the user to setup the functionality of the STB has happened during cycle 5,000-10,000. An ITV event is triggered when the user wants to browse the player files during watching a basketball game. After the ITV application has been activated, CPU requests a large amount of bandwidth instantly to process the application. The user then paused the TV program temporarily during cycle 21,000-31,000 to browse the information offered by the broadcaster.

2) Simulation Results

To examine the QoS performance, we look at the fulfillment of bandwidth allocation first. The latency of CPU is assessed later. For round-robin controllers shown in Figure 16(a) and Figure 16(b), the limited bandwidth provided by SIG-RR-MIS-I and SIG-RR-MIS-II obviously cannot fulfill the requirement of STB SoC. As for MUL-RR-MIS-II shown in Figure 16(c), although the overall bandwidth utilization is much better, the bandwidth allocation for each PU is unacceptable. As mentioned above,

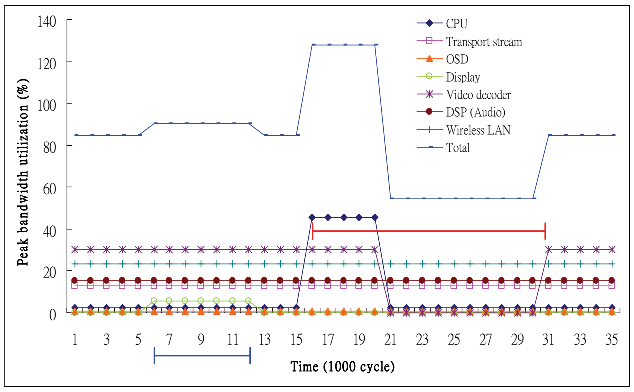
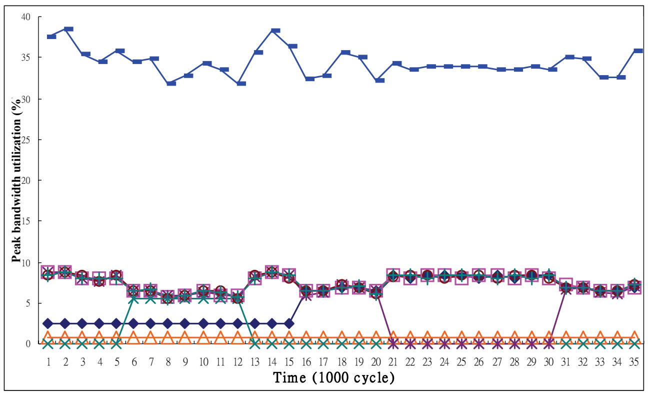


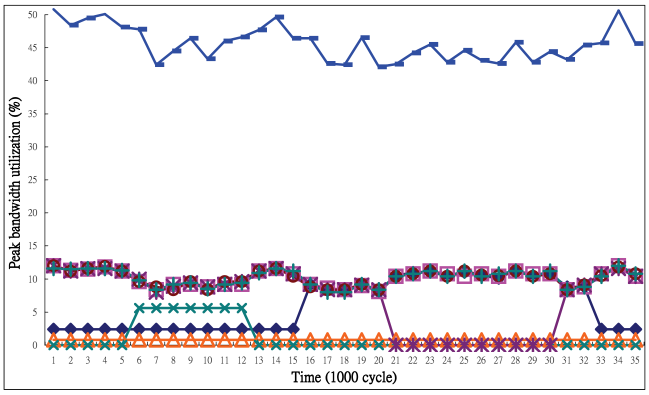
Figure 15. Bandwidth requirement of each PU in the STB SoC

Table 5. Configuration of PUs in the STB system

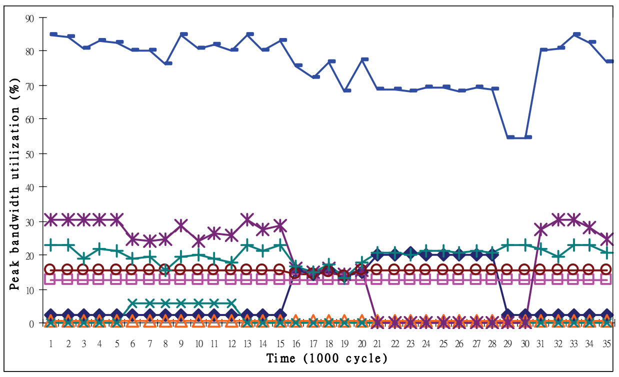
PU	Fixed-priority	Quality-aware
CPU	1st priority	Latency-sensitive
Transport stream	2nd priority	Bandwidth-sensitive
DSP	3rd priority	Bandwidth-sensitive
OSD	4th priority	Bandwidth-sensitive
Video decoder	5th priority	Bandwidth-sensitive
Display	6th priority	Bandwidth-sensitive
Wireless LAN	7th priority	Don't-care



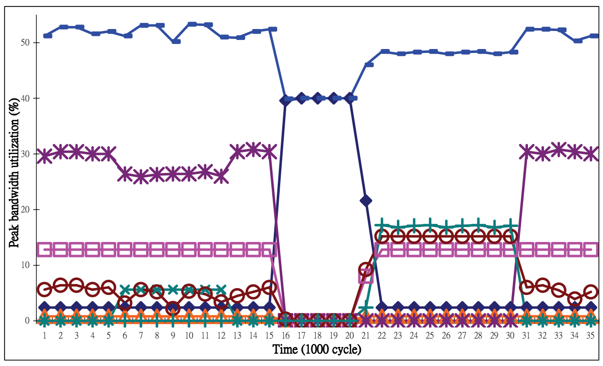
(a)



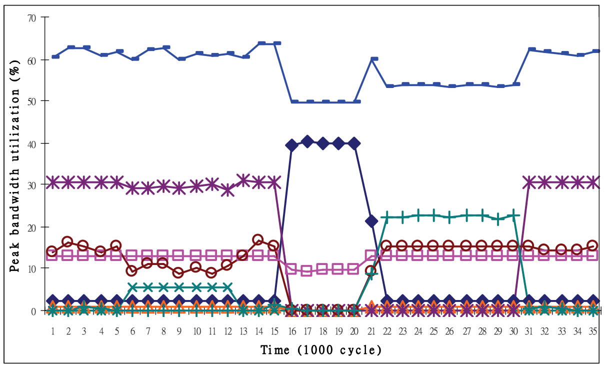
(b)


Peak bandwidth utilization (%)
Time (1000 cycle)

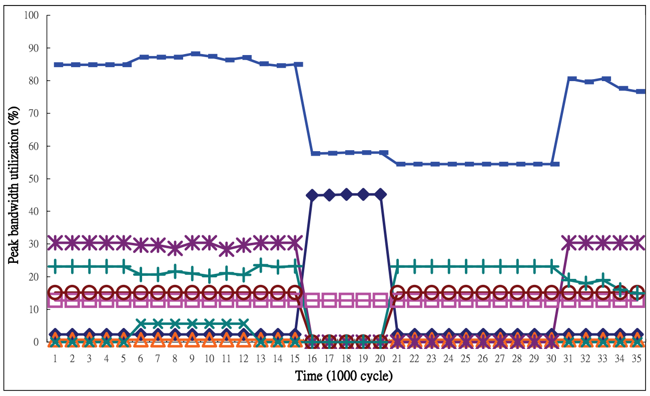
(c)


Peak bandwidth utilization (%)
Time (1000 cycle)

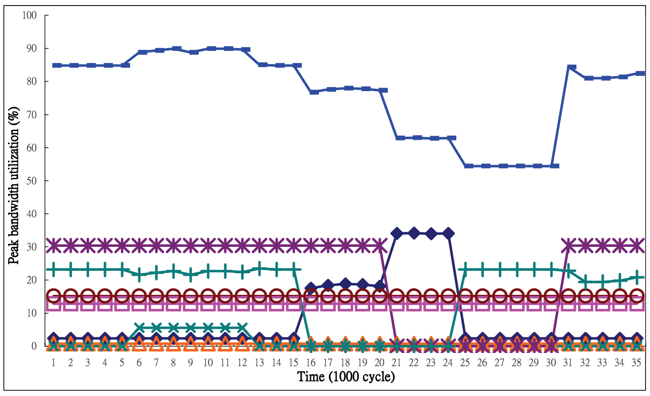
(d)


Peak bandwidth utilization (%)
Time (1000 cycle)

(e)



(f)



(g)

Figure 16. Bandwidth utilization of (a) SIG-RR-MIS-I, (b) SIG-RR-MIS-II, (c) MUL-RR-MIS-II, (d) SIG-FP-MIS-I, (e) SIG-FP-MIS-II, (f) MUL-FP-MIS-II, and (g) QA-MIS-II

round-robin controllers evenly allocate total bandwidth to each PU. Thus, both OSD and ITV events may result in quality degradation of the broadcasting program. Take the ITV event for example, CPU requests a large bandwidth instantly and some bandwidth for the video decoder and the display unit is allocated to CPU. Therefore, the guaranteed bandwidth requirements of these two PUs are ruined.

As for fixed-priority controllers shown in Figure 16(d) and Figure 16(e), the single-channel fixed-priority controllers can provide higher bandwidth utilization than single-channel round-robin controllers can. This is because they allow high priority PUs to access DRAM uninterruptedly and hence can avoid bandwidth loss due to bus handover and frequent row reopening. However, the provided bandwidth is still not enough. For example, no bandwidth is allocated to wireless LAN controller during the normal operation period. The bandwidth provided by MUL-FP-MIS-II is much higher. However, as indicated in Figure 16(f), the bandwidth allocation problem still exists. During the ITV event, CPU takes a large portion of bandwidth. This severely degrades the quality of the broadcasting program and is therefore unacceptable.

Figure 16(g) illustrates the simulation result of quality-aware controller. As we can see, the bandwidths allocated for all bandwidth-sensitive PUs are well guaranteed. When ITV event happens, the bandwidth for wireless LAN controller is taken first. The quality of the broadcasting program remains unchanged when the user is still watching the program. After the TV program has been paused, CPU takes the spared bandwidth released by the video decoder for the ITV program. In addition, accesses from the wireless LAN controller can be served as much as possible during this period.

In Figure 17, CPU access latency is measured separately when CPU operates during the normal operation period and the ITV event. First we take a look at the normal operation period. As expected, round-robin controllers have the longest access latencies compared to other controllers due to the fair scheduling, which apparently cannot fit CPU's low latency requirement. By granting accesses from CPU with the highest priority, fixed-priority controllers can effectively reduce the access latencies. Among these fixed-priority controllers, MUL-FP-MIS-II has the longest CPU access latency because its high bandwidth utilization causes serious SDRAM command and data bus congestion. The congestion problem is effectively eliminated in

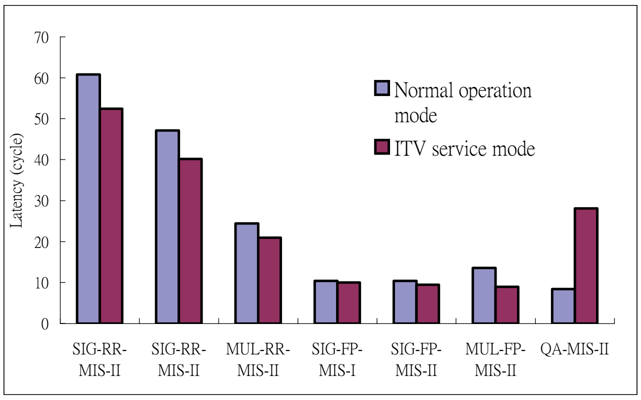


Figure 17. CPU access latency in different operation mode of each SDRAM controller

QA-MIS-II. With preemptive and CAI services, CPU access latency of QA-MIS-II is reduced by about 19% and 37% compared to SIG-FP-MIS-II and MUL-FP-MIS-II respectively.

During the ITV event, CPU access latencies of round-robin controllers are lower than those latencies of round-robin controllers during normal operation mode, since the video decoder is paused in cycle 21,000-30,000. Similarly, fixed-priority controllers also have shorter CPU access latency during the ITV event. This is because the high request rate of CPU blocks other PUs to access DRAM and hence preserves most DRAM resources for CPU. In contrast, the latency of QA-MIS-II is longer than that of fixed-priority controllers since the bandwidth requirement of the ITV applications is taken as don’t care type. This is acceptable because users are often less sensitive to the execution speed of ITV applications.

3.6 Summary

We have presented a multi-layer, quality-aware SDRAM controller for multimedia platform SoCs. The layered architecture is motivated by the awareness of that not every system needs the same requirement of memory usage. Therefore, we well partition the functionality of a memory controller into proper layers such that designers have the flexibility to adopt the best fitting layers for various applications. By appropriately categorizing channels into three types, QAS is able to provide the best DRAM services including short access latency and guaranteed bandwidth for each type of channels. DRAM bandwidth utilization is improved by the support of parallel access of each bank within SDRAM and the ability to issue every DRAM command at the earliest time available. The configurability of MIS, based on the shared-state FSM design, can alleviate the burden for system designers by rapid integration of SDRAM subsystem. Some recently developed systems, especially those for portable applications, have power management with the ability to control the system clock frequency in adjusting system performance to just fit to the requirement. Programmability of DRAM control latencies enables the power management to dynamically lower the clock frequency of MIS.

The results of STB experiment show that the access latency of the latency-sensitive channel can be reduced by 37% and 65% compared to conventional multi-channel fixed-priority and round-robin controllers respectively. Furthermore, the memory bandwidths can be precisely allocated to bandwidth-sensitive channels with a high degree of control and no bandwidth-sensitive channel suffers starvation in all simulated STB events. In summary, the presented memory controller can achieve high DRAM utilization while still meeting different memory access requirements of bandwidth and latency.

4. CONCLUSION

Memory represents a critical driver in terms of cost, performance and power for embedded systems. To address this problem, a large variety of memory technologies and memory access managements have been proposed. On one hand the application is characterized by a variety of access patterns. On the other hand, new memory devices and organizations provide a set of features. To find the best match between the application characteristics and the memory organization features, the designer needs to explore different memory configurations in combination with different design architectures. Furthermore, the growing number of cycles required for memory accesses also caused designers to implement latency-tolerance techniques such as prefetching and out-of-order execution. Farther in the future, new memory-centric architectures, tools and design methodologies may be developed specifically to improve the cost, power and performance of memory systems. Most of these solutions will eventually be used synergistically to meet the severe requirements of embedded systems.

Due to the advance of hardware, more complex algorithms and systems are now investigated or already available to promote new functionality or better services. These complex designs also create a new requirement of memory optimization. For example, more algorithms explore bit-level optimization for better performance. While in the past the main effort has been to optimize designs at word level or sub-word level, new and unexplored degrees of freedom become available when design optimization is explored at the bit level[70,71,72]. In addition to bit-level processing, non-multiple of eight bits per data sample is widely adopted in video applications, such as high quality television signal and high profiles of H.264/AVC[73]. This size of data sample also creates design challenges of memory system, which is traditionally used for data sample with a size of multiple of eight.

While we conjecture that algorithmic designs will remain based on human intuition and ingenuity, we believe that the parameter tuning and search of an optimal architecture in a restricted domain can be at least methodized or partially automated. The formulation of rigorous theories and optimization techniques for memory system designs is an exciting area for future research.

REFERENCES

1. G. Goossens, et al, "Synthesis of flexible IC architectures for medium throughput real-time signal processing," *J. VLSI signal processing*, vol.5, no.4, Kluwer Academic Publishers, Boston, 1993.

2. Denali Memory Report, vol 1, issue 4, May 2002.
3. T. H. Meng, B. Gordon, E. Tsern, and A. Hung, "Portable video-on-demand in wireless communication," *in Proc. of the IEEE*, Vol.83, No.4, pp.659-680, Apr. 1995.
4. V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," in *Proc. ICCAD*, pp.384-390, Nov. 1994.
5. F. Catthoor et al. *Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design*. Kluwer Academic Publishers, 1998.
6. D. Burger, J. R. Goodman, and A. Kagi, "Limited bandwidth to affect processor design," *IEEE Micro*, 17(6): pp. 55--62, Nov./Dec. 1997.
7. P. R. Panda, N. Dutt, and A. Nicolau, *Memory issues in embedded systems-on-chip: optimizations and exploration*, Kluwer Academic Publishers, Boston, 1999.
8. C. Natarajan, B. Christenson, and F. Briggs, "A study of performance impact of memory controller features in multi-processor server environment," *in Proc. of the 3rd workshop on Memory performance issues*, pp. 80-87, 2004.
9. D. A. Patterson, "Latency lags bandwidth", *Communication of the ACM*, vol. 47. no. 10, pp. 71-75, Oct. 2004.
10. K. Kilbuck, "FCRAM 101 Part 1: Understanding the Basics", CommsDesign, 2002. [Online]. Available: http://www.commsdesign.com/printableArticle/?articleID=16504491
11. N. C. C. Lu, "Emerging technology and business solutions for system chips," *ISSCC Dig. Tech. Papers*, pp.25-31, Feb. 2004.
12. A. K. Khan, *et al.*, "A 150-MHz graphics rendering processor with 256-Mb embedded DRAM", *IEEE J. Solid-State Circuits*, vol. 36, no. 11, pp. 1775-1784, Nov. 2001.
13. M. Takahashi, *et al.*, "A 60-MHz 240-mW MPEG-4 videophone LSI with 16-Mb embedded DRAM", *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1713-1721, Nov. 2000.
14. B. Dipert, "Embedded Memory: The All Purpose Core", EDN Magazine, Mar. 1998. [Online]. Available: http://www.ednmag.com/reg/1998/031398/06cs.cfm
15. L. Benini, A. Macii, and M. Poncino, "Energy-aware design of embedded memories: a survey of technologies, architectures and optimization techniques", ACM Trans. Embedded Computing Systems, vol. 2, no. 1. pp. 5-32, Feb. 2003.
16. K. Nii, et al., "A 90-nm low-Power 32-kB embedded SRAM with gate leakage suppression circuit for mobile Applications", *IEEE J. Solid-State Circuits*, vol. 39, no. 4, pp. 684-692, Apr. 2004.
17. W. Leung, F. C. Hsu, and M. E. Jones, "The ideal SoC memory: 1T-SRAM" Proc. IEEE ASIC/SoC Conf., pp. 32-36, 2000.
18. P.C. Fazan, et al., . "A simple 1-transistor capacitor-less memory cell for high performance embedded DRAMs", *Proc. IEEE CICC*, pp.99-102 , 2002
19. NEC, "New ASIC Process Technology Makes Embedded DRAM Practical Choice For High-Performance Applications", [Online]. Available: http://www.necel.com/en/process/pdf/eDRAMwhitepaper3.7.pdf
20. SST, "SuperFlash EEPROM technology", [Online]. Available: http://www.sst.com/downloads/tech_papers/701.pdf
21. K. Ayukawa, T. Watanabe and S. Narita, "An access-sequence control scheme to enhance random-access performance of embedded DRAM's," *IEEE J. Solid-State Circuits*, vol. 33, no. 5, pp. 800-806, May 1998.
22. T. Watanabe et al., "Access optimizer to overcome the future walls of embedded DRAMs in the era of systems on silicon," in *Proc ISSCC99*, pp. 370 -371, 15-17 Feb. 1999.

23. S. A. McKee et al, “Experimental implementation of dynamic access ordering,” in *Proc. of the 27th Hawaii International Conference on System Sciences,* pp. 431-440, Jan. 1994.
24. S. A. McKee and Wm. A. Wulg, “A memory controller for improved performance of streamed computations on symmetric multiprocessors,” in *Proc IPPS '96*, pp. 159-165.
25. S. I. Hong et al., “Access order and effective bandwidth for streams on a Direct Rambus memory,” in *Proc. of the 5th HPCA*, pp. 80-89, Jan. 1999.
26. P. R. Panda, N. D. Dutt, and A. Nicolau, “Incorporating DRAM access modes into high-level synthesis,” *IEEE Trans. CAD*, vol. 17, no. 2, pp. 96-109, Feb. 1998.
27. A. Khare, P. R. Panda, N. D. Dutt, and A. Nicolau, “High-level synthesis with synchronous and RAMBUS DRAMs,” in *Proc. SASIMI '98*, pp.186-193, 1998.
28. P. R Panda and N. D. Dutt, “Low-power memory mapping through reducing address bus activity,” *IEEE Trans. VLSI Syst*, vol. 7, pp. 309-320, Sept. 1999.
29. M. Winzker, P. Pirsch and J. Reimers, “Architecture and memory requirements for stand-alone and hierarchical MPEG2 HDTV-decoders with synchronous DRAMs,” in *Proc ISCAS95*, pp. 609 -612, Jan. 1995.
30. T. Gleerup et al., “Memory architecture for efficient utilization of SDRAM: a case study of the computation/memory access trade-off,” in *Proc. of the 8th International Workshop on Hardware/Software Codesign*, pp. 51-55, 2000.
31. H.-K. Chang and Y.-L. Lin, “Array allocation taking into account SDRAM characteristics,” *in Proc. ASP-DAC*, pp. 497-502, Jan. 2000.
32. H. Schmit and D. E. Thomas, Jr., “Address generation for memories containing multiple arrays,” *IEEE Trans. CAD*, vol. 17, issue 5, pp.377 -385, May 1998.
33. A. Jantsch, et al., “Hardware/software partitioning and minimizing memory interface traffic,” *Proc. of the EuroDAC*, pp.226-231, 1994.
34. N. Chang, K. Kim, J. Cho and H. Shin, “Bus encoding for low-power high-performance memory systems,” in *Proc.* DAC2000, pp. 800-805, Jun 2000.
35. W.-C. Cheng and M. Pedram, “Power-optimal encoding for DRAM address bus,” in *Proc. International Symposium on Low Power Electronics and Design*, pp. 250-252, 2000.
36. C. L. Su, C. Y. Tsui, and A. M. Despain, “Saving power in the control path of embedded processors,” *IEEE Design and Test of Computers*, vol. 11, pp. 24-30, 1994.
37. M. R. Stan and W. P. Burleson, “Bus-invert coding for low-power I/O,” *IEEE Trans. VLSI Syst.*, vol. 3, no. 1, pp. 49-58, Mar. 1995.
38. Y. Shin, S. Chae, and K. Choi, “Reduction of bus-transitions with partial bus-invert coding,” *IEE Electronics Letters*, vol.34, no. 7, pp.642-643, Apr. 1998.
39. S. Hong, T. Kim, U. Narayanan, and K. S. Chung, “Decomposition of bus-invert coding for low power I/O,” *J. Circuits, Syst., Comput.*, vol. 10, pp. 101-111, 2000.
40. M. Mamidipaka, D. Hirschberg, and N. Dutt, “Low power address encoding using self-organizing lists,” in *Proc. ISLPED'01*, pp. 188-193, Aug. 2001.
41. P. With, P. Frencken, and M. Schaar-Mitrea, “An MPEG decoder with embedded compression for memory reduction,” *IEEE Trans. Consumer Electron.*, vol. 44, pp. 545-555, Aug. 1998.
42. T. Y. Lee, “A new frame-recompression algorithm and its hardware design for MPEG-2 video decoders,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 529-534, June 2003.
43. S.-B. Ng, *Lower Resolution HDTV Receivers*, US patent 5262854, Nov. 1993.
44. W. Zhu, K. H. Yang, and F. A. Faryar, “A fast and memory efficient algorithm for down-conversion of an HDTV bitstream to an SDTV signal,*” IEEE Trans. Consumer Electron.*, vol. 45-1, pp. 57-61, Feb. 1999.

45. L. Benini and G. D. Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Academic Publishers, 1998.
46. A. H. Farrahi, G. E. Téllez, and M. Sarrafzadeh, "Memory segmentation to exploit sleep mode operation," in *Proc DAC95*, pp.36-41, June 1995.
47. H. Heske, Mobile RAMs can help save power, Portable Design Magazine, July 2002. [Online]. Available: http://www.electronicsforu.com/electronicsforu/articles/hits.asp?id=369
48. R. Goering, "Philips design team wins EDAC award," EEdesign, May 30, 2002.
49. S. Dutta, R. Jensen, and A. Rieckmann, "Viper: A multiprocessor SoC for advanced set-top box and digital TV systems," *IEEE Des. Test. Comput*., vol. 18, no. 5, pp. 21-31, Sept.-Oct. 2001.
50. G. Martin and H. Chang, *Winning the SoC Revolution: Experiences in Real Design*, Kluwer Academic Publishers, Boston, Jun. 2003.
51. B. Furht, "Multimedia systems: an overview," *IEEE Multimedia*, vol. 1, no. 1, Spring 1994, pp. 47-59.
52. J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed., Morgan Kaufmann Publishers, San Francisco, 2002.
53. A. Cataldo, MPU designers target memory to battle bottlenecks, EE Times, (10/19/01). [Online]. Available: http://www.siliconstrategies.com/ story/OEG20011019S0125
54. R. C. Schumann, "Design of the 21174 memory controller for DIGITAL personal workstations," *Digital Technical Journal*, vol. 9, no. 2, pp. 57-70, 1997.
55. J. Carter et al., "Impulse: Building a smarter memory controller," in *Proc. HPCA 1999*, pp. 70-79, Jan. 1999.
56. S. Rixner, et al., "Memory access scheduling," in *Proc. ISCA 2000,* Vancouver, Canada, June 2000, pp. 128-138.
57. T. Takizawa and M. Hirasawa, "An efficient memory arbitration algorithm for a single chip MPEG2 AV decoder," *IEEE Trans. Consumer Electron*., vol. 47, no.3, pp. 660-665, Aug. 2001.
58. J. Corbal, R. Espasa, and M. Valero, "Command vector memory systems: High performance at low cost," in *Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques*, pp. 68-77, Oct. 1998.
59. Sonics, Efficient Shared DRAM Subsystems for SoCs, 2001. [Online]. Available: http:// www.sonicsinc.com/sonics/products/memmax/productinfo/docs/DRAM_Scheduler.pdf
60. Sonics, SoCCreator Guide Design Flow. [Online]. Available: http://www.socworks.com/ socworks/support/documentation/html/
61. K. Lahiri, A. Raghunathan, and G. Lakshminarayana, "LOTTERYBUS: a new high-performance communication architecture for system-on-chip designs," in *Proc. Design Automation Conference*, pp.15-20, Jun. 2001.
62. F. J. Harmsze, A. H. Timmer, and J. L. van Meerbergen, "Memory arbitration and cache management in stream-based systems," in *Proc. DATE 2000*, Mar. 2000, pp. 257-262.
63. Denali Software Inc., Databahn product information, [Online]. Available: http://www.denali.com/products_databahn_dram.html.
64. K.-B. Lee, T.-C. Lin, and C.-W. Jen, "An efficient quality-aware memory controller for multimedia platform SoC," *IEEE Trans. Circuits Syst. Video Technol*., vol. 15, pp. 620-633, May. 2005.
65. K.-B. Lee and C.-W. Jen, "Design and verification for configurable memory controller - Memory interface socket soft IP," *Journal of the Chinese Institute of Electrical Engineering*, vol. 8, no. 4, pp.309-323, 2001.
66. ARM, Inc. PrimeXsys Platforms. [Online]. Available: http://www.arm.com/armtech/ PrimeXsys?OpenDocument

67. Bill Cordan, “An efficient bus architecture for system-on-chip design,” *IEEE Custom Integrated Circuits*, San Diego, USA, May 1999, pp. 623 -626.
68. S. Hosseini-Khayat and A.D. Bovopoulos, “A simple and efficient bus management scheme that supports continuous streams,” *ACM Trans. Computer Systems*, vol. 13, no. 2, pp. 122-140, 1995.
69. Micron Technology, Inc. mt48lc16m16a2 256Mb SDRAM, Jan 2003. [Online]. Available: http://www.micron.com/products/datasheet.jsp?path=/DRAM/SDRAM&fileID=10
70. M.-Y. Chiu, K.-B. Lee, and C.-W. Jen, “Optimal data transfer and buffering schemes for JPEG2000 encoder,” in *Proc. SIPS 2003*, pp.177-182, Aug. 2003.
71. K.-B. Lee et al., “Optimal frame memory and data transfer scheme for MPEG-4 shape coding,” *IEEE Trans. Consumer Electron.*, vol. 50, no.1, pp. 342-348, Feb. 2004.
72. A. Erturk and S. Erturk, “Two-bit transform for binary block motion estimation,” *IEEE Trans. Circuits Syst. Video Technol*., vol. 15, Issue 7, pp. 938-946, July 2005.
73. G. J. Sullivan, P. Topiwala, and A. Luthra, “The H.264/AVC advanced video coding standard: overview and introduction to the fidelity range extensions,” in *Proc. SPIE*, Denver, Aug. 2004.

Chapter 5

EMBEDDED SOFTWARE

Tai-Yi Huang*, Shiao-Li Tsao[†], Le-Chun Wu[#], Edward T.-H Chu*, and Ko-Yun Liu*

*National Tsing-Hua University
[†] National Chiao-Tung University
[#]National Taiwan University

1. INTRODUCTION

The advancement of semiconductor manufacturing technology makes it practical to place a traditional board-level embedded system on a single chip. The evolvement of system-on-chip (SoC) techniques presents new challenges on integrated circuit (IC) designs as well as embedded software and systems. A SoC system usually has limited hardware resource or functionality such as battery capacity, slower processors, and small memory, which induce the complexity of embedded software design. Traditional system software cannot simply be directly deployed on a SoC system to fully utilize its capabilities without considerable modifications in design and implementation. Software development tools, such as compilers, linkers, loaders, assemblers, debuggers, and simulators, have therefore become an integral part of the SoC system design. The requirement and design methodology of these tools are quite different from those on the general-purpose computing systems. In addition, conventional approaches to developing software usually cannot proceed until the hardware is ready and fully tested. In contrast, hardware/software co-design becomes a crucial step in the development of SoC embedded software. It significantly speeds up the design and implementation process of hardware and software of an embedded system. In this chapter, we first discuss low-power scheduling of embedded software, which is a core issue in an embedded system design. It includes an in-depth and broad introduction on low-power task scheduling

Y.-L.S. Lin (ed.), Essential Issues in SOC Design, 119–175.
© 2006 *Springer.*

and device scheduling. We later investigate the development framework for device drivers and hardware/software co-design methodology. Finally, we look into the topics in compiler and software development toolchains.

Modern embedded systems, such as sensors and portable and wireless devices, are often powered by batteries. Due to its limited capacity of energy, the problem of reducing energy consumption has become a main concern in embedded system design. In addition, tasks running on such a system often impose real-time constraints that require a response to be returned before a deadline. A task that fails to complete its execution before its deadline results in the failure of the task and the whole system. Examples of power-aware real-time embedded systems include, but are not limited to, cell phones, digital cameras, and sensor devices. A voice package must be sent within a time period to provide good-quality communication and avoid jittered delay. The operation of a sensor network requires that the sensed data be returned before a deadline to complete a multi-sensor decision. Data that arrives after the decision is made presents no value at all in many scenarios. Without energy-hungry components such as disks and CD-ROMs, the processor constitutes a major source of energy consumption on an embedded device. For this reason, the problem of low-power real-time task scheduling that arranges task execution for minimum energy consumption has received a lot of attentions recently.

Dynamic Voltage Scaling (DVS) is a commonly-used technique for reducing processor energy consumption. However, reducing the supply voltage of a processor leads to a linear extension of the execution time of a task. Therefore, the issue of minimizing total energy consumption without violating any real-time constraint becomes the main challenge in the design of a low-power real-time system. A real-time DVS algorithm makes use of slack time to speed down the processor and reduce its energy consumption. Slack time is an amount of time a job can be delayed without causing any job to miss its deadline. A number of real-time DVS algorithms have been developed over the past few years to calculate available slack time and speed down the processor. Section 2 investigates several typical real-time DVS algorithms and its latest development.

I/O subsystems have recently become a major source of energy consumption in embedded systems (Simunic et al., 2001; Choi et al., 2002; HP Lab and Alto, 2003). Dynamic Power Management (DPM) is gaining importance due to constraints on power budgets of these systems. DPM puts idle devices into a low-power and low-performance state to save energy. However, because it takes extra time and energy to change power states of a device, whether a DPM policy switches on or off a device requires careful analysis and calculation on each device's idle interval. Many algorithms had been proposed for the prediction of a device's idle interval. The experimental results show that, when idle intervals of devices are accurately predicted, DPM can significantly reduce the energy consumption of devices. Because real-time operating systems (RTOS) are used in many modern embedded systems, the problem of applying DPM without violating real-time constraints also needs to be addressed. Finally, there exists a demand for a hybrid power management that integrates both DVS and DPM to reduce system-wide energy consumption. Section 3 gives an overview and classification of latest DPM technology.

Device drivers play an important role in the embedded software design because an embedded system is generally equipped with various peripheral devices, and handles external events through input/output (I/O) channels. Device drivers can significantly impact performance and real-time properties of an embedded system. The path of an interrupt service in a system with OS behaves extremely different from the one in a system without OS. How to improve the efficiency and interrupt latency of a device driver under different system configurations becomes an important issue (Jerraya et al., 2003; Labrosse, 2002; Li and Yao, 2003). On the other hand, hardware/software co-design becomes more and more important in device driver development. An efficiency co-design methodology can considerably reduce the development and debugging time of device drivers, but there are still a number of challenges needed for further research. Section 4 presents the characteristics, operations and design issues of a device driver for an embedded system. The section first summarizes characteristics of device drivers and the differences between a device driver for a general-purpose system and one for an embedded system. It is then followed by the discussion of a list of important issues in hardware/software (HW/SW) co-design flow, such as portability, testability and so on.

Software development tools have long been used extensively for construction, debugging, and testing of software code. A well designed and easy-to-use toolchain can help developers to cut down the software implementation time significantly and therefore reduce the total design cost. While the structures and functionalities of toolchains for embedded systems are essentially no different from those for general-computing systems, due to the limited resources of embedded systems, different sets of challenges are presented when designing embedded tools. Among all the software toolchain components, the compiler is probably considered the most important. Besides relieving the software developers of the burden of assembly code writing, the compiler is especially pivotal in making sure the software code meets the system requirements. Unlike general-purpose compilers where run-time performance is often the foremost concern when generating and optimizing code, because of hardware resource constraints and time-to-market pressure, other factors such as power consumption, code size, and retargetability are as important as performance when generating embedded code. Section 5 begins with an overview of embedded software development tools and their basic structures, followed by discussions of several important research issues for embedded compilers.

2. LOW-POWER TASK SCHEDULING

The technique of dynamic voltage scaling (DVS) is considered the most efficient and important approach for reducing the energy consumption of processors. In this section, we first briefly explain DVS and its real-time extension. We next describe a set of mechanisms to utilize unused CPU bandwidth to speed down the processor. We finally present and classify a list of real-time DVS algorithms.

2.1 Real-Time Dynamic Voltage Scaling

The power consumption of a processor is dominated by its dynamic power dissipation, denoted by P. That is,

$$P = aC_f V_d^2 S$$

where a is the average activity factor, C_f is the effective switched capacitance, V_d is the supply voltage, and S is the processor speed. Furthermore, the processor speed S is nearly linearly related to the supply voltage V_d as

$$S = k\frac{(V_d - V_t)^\alpha}{V_d}$$

where k is a constant specific to a given technology, V_t is the threshold voltage, and α is the velocity saturation index, $1 < \alpha < 2$. DVS makes use of these characteristics to reduce P, the power consumption of a processor, cubically by lowering V_d, its supply voltage, for most processors.

However, reducing the supply voltage of a processor leads to a linear extension of the execution time of a task. Because the total energy consumption of a task equals to the multiplication of the power consumption and the execution time of the task, we only reduce the energy consumption of a task quadratically when scaling down the supply voltage. On the other hand, a real-time system requires completing each task before its deadline. Therefore, the issue of minimizing total energy consumption without violating any real-time constraint becomes the main challenge in the design of a low-power real-time system.

2.2 Slack Time Usage

Slack time is an amount of time a job can be delayed without causing any job to miss its deadline. Slack time is available when the sum of task utilizations is less than 100% or a task completes earlier than its worst-case execution time (WCET). The former available time is called static slack time as it can be calculated off-line. The latter is called dynamic slack time which cannot be determined until run-time. A real-time DVS algorithm makes use of slack time to speed down the processor and reduce its energy consumption.

We divide the discussion of slack time usage for reducing processor speed into two parts: static slack usage and dynamic slack usage. The common methods used to utilize static slack are minimum constant speed, priority-monotonic, essential interval, and EDF transformation. The common methods used to utilize dynamic slack are NTA stretching, priority-based slack stealing, work-demand analysis, and utilization update.

2.2.1 Static slack time usage

Minimum constant speed

The most popular way of utilizing static slack time is to calculate the total workload and, based on this workload, determine a minimum processor speed. Every task is executed constantly at this speed to ensure that no task misses its deadline. The calculation of this minimum constant speed varies on different algorithms. We will describe the details and differences later when needed.

Priority-monotonic

This method applies to a system consisting of periodic tasks. It assigns processor speeds to tasks in monotonic priority order. A task with a higher priority is assigned with a same or faster processor speed than a task with a lower priority. That is, the processor speed for a task τ_i is not slower than the speed for τ_j, if the priority of τ_i is the same or higher than τ_j. To determine the processor speed for τ_i, we first identify its critical period (*i.e.*, the one that starts at a critical instant). We next calculate the utilization at each interval ending with slack time in this critical period. Let w denote the workload of an interval. That is, w is set to the sum of the execution time of higher-priority jobs than τ_i in this interval. The utilization of this interval is defined to be the value of w divided by the length of the interval. We define the critical interval of τ_i as the interval of the maximum utilization. The processor speed for τ_i, denoted by S_i, is set to the utilization of its critical interval. If this speed is slower than τ_{i+1}'s speed, we simply set it to τ_{i+1}'s speed, in order to comply with the monotonic order.

Figure 1 gives an example to illustrate the priority-monotonic approach. There are 3 periodic tasks, τ_1, τ_2, and τ_3, that are released at $t = 0$. The critical period of τ_1 is [0, 5] and its critical interval is also [0, 5]. The workload w of this interval is 3. Thus, S_1 is set to 0.6. The critical period of τ_2 is [0, 7]. The critical interval is [0, 5] and its utilization is 0.8. Accordingly, S_2 is set to 0.8. The critical period of τ_3 is [0, 21]. The intervals ending with slack time are [0, 10], [0, 14], and [0, 20] and their utilizations are 0.9, 0.86, and 0.8, respectively. Therefore, the critical interval is [0, 20] and S_3 is set to 0.8. To comply with the monotonic order, we set $S_1 = S_2 = S_3 = 0.8$.

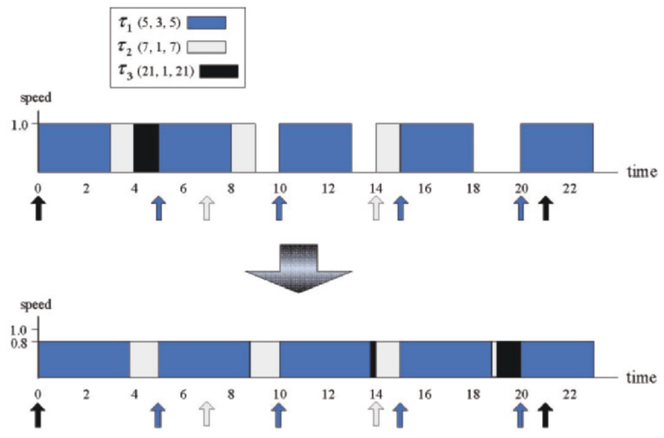


Figure 1. The usage of static slack time in the priority-monotonic approach

Essential interval

The essential-interval approach treats each job as an independent job. Each job J_i is denoted by (r_i, e_i, d_i), where r_i is its release time, e_i is its WCET at the maximum processor speed, and d_i is its absolute deadline. The essential interval of J_i is defined to be $[r_s, d_i]$, where r_s is the release time of a job J_s. J_s is selected such that $s \leq i$, $r_s \leq r_i < d_s$, and $[r_s, d_i]$ has the maximum utilization among all possible s. We then set the processor speed of this essential interval to be its utilization in order to fully utilize available static slack time. Figure 2 shows a 4-job example where J_1 has the highest priority and J_4 has the lowest priority. The essential interval of J_4 is $[r_s, 20]$, where s can be 2 or 3. The utilization of the interval $[r_2, 20]$ is 0.7 and the utilization of the interval $[r_3, 20]$ is 0.47. As a result, the essential interval of J_4 is [10, 20] and its processor speed is set to 0.7.

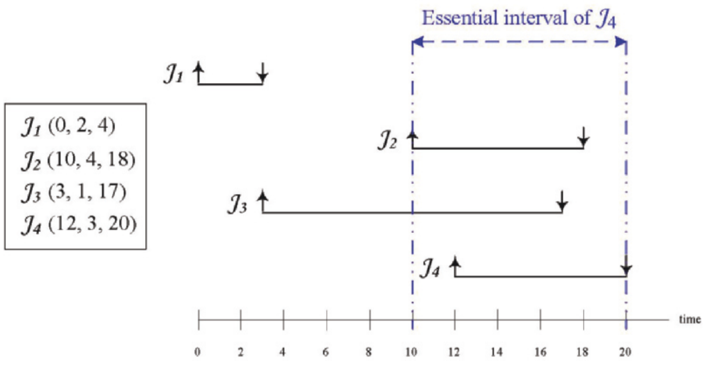


Figure 2. The essential interval of J4

EDF transformation

This technique utilizes available slack time by first transforming a fixed-priority set of jobs into a conforming set of EDF jobs and scheduling them by a known optimal DVS-capable EDF scheduling algorithm. If the fixed-priority assignment of jobs is the same as an EDF schedule, we simply schedule them with the optimal EDF algorithm. Figure 3(a) shows a case of such a set of jobs, J_1, J_2, and J_3, that have the same priority order in both fixed-priority assignments and EDF scheduling. If a set of jobs have a different priority order than EDF scheduling but there is no interference between these jobs, we still schedule them with the optimal EDF algorithm. Figure 3(b) shows a case of such jobs where J_1 has a fixed higher priority than J_3 but J_3 has a deadline earlier than the release time of J_1. For other sets of jobs, their deadlines are modified to create an EDF schedule without violating original priority assignments. One kind of transformation is shown in Figure 3(c) to set the deadlines of J_2 and J_3 at the same time as the release time of J_1 in Figure 3(d). The transformed set of jobs are scheduled by the optimal DVS-capable EDF algorithm to make use of available static slack time.

2.2.2 Dynamic slack time usage

NTA stretching

The approach of NTA stretching utilizes available static and dynamic slack time to execute tasks at a constant reduced speed between the current

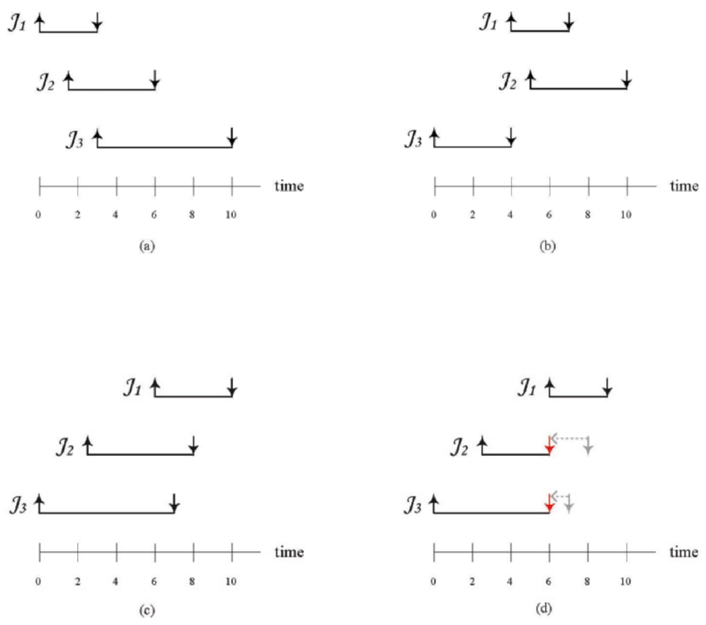


Figure 3. The approach of EDF transformation

instant and the earliest deadline. Let t_c denote the current instant and t_d denote the next task arrival. Let w denote the worst-case workload in [t_c, t_d]. The available slack time at t_c is equal to $(t_d - t_c - w)$. This approach executes jobs in the interval of [t_c, t_d] at the speed of $w/(t_d - t_c)$ to fully utilize available slack time. For example, in Figure 4, τ_1 and τ_2 are released at $t = 0$, and the next task arrival is at $t = 10$. We set the speed at $t = 0$ to $(3 + 5)/(10 - 0) = 0.8$ to utilize static slack time. When τ_1 completes earlier at $t = 2$, we recalculate the available slack time to take dynamic slack into account and set the speed to $5/(10 - 2) = 0.625$.

Priority-based slack stealing

The priority-based stack-stealing approach calculates available slack time at the completion of each job. If a job completes earlier than its WCET, the available dynamic slack time will be assigned to a released job and this job will be executed at a reduced processor speed. When the assignment of slack time follows a priority-based manner, we call such a method a priority-based stack-stealing algorithm.

Figure 5 shows an example of 3 tasks all released at $t = 0$. Their executing speed is set to 1 initially. When τ_1 completes earlier at $t = 0.5$, as shown in Figure 5(a), the 0.5 dynamic slack is assigned to τ_2, a released job

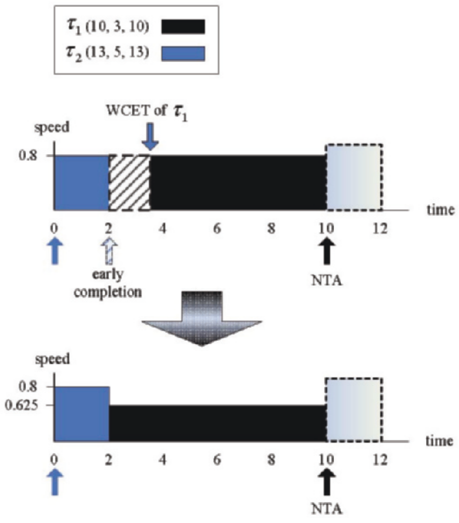


Figure 4. The approach of NTA stretching

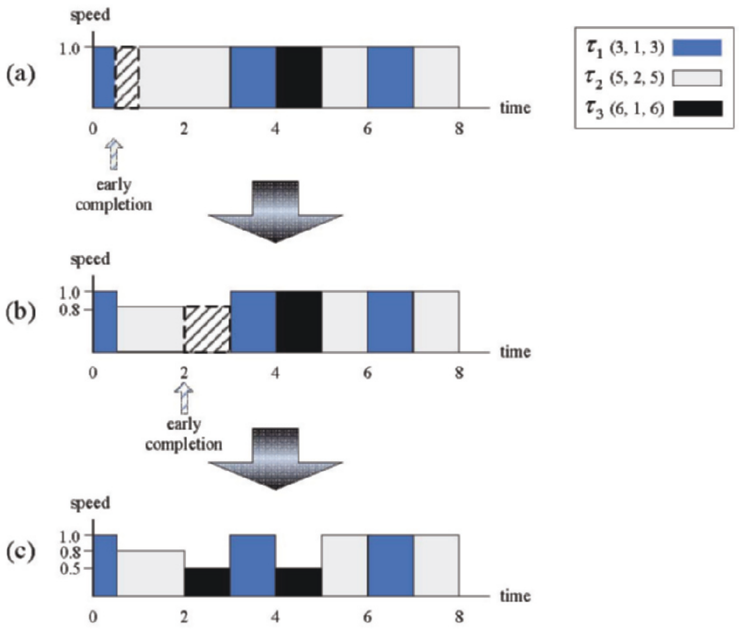


Figure 5. The priority-based slack-stealing approach

with the next lower priority. τ_2 makes use of this dynamic slack to reduce its processor speed to 2/2.5 = 0.8. When τ_2 completes earlier at $t = 2$, as shown in Figure 5(b), the 1 dynamic slack is assigned to τ_3. τ_3 reduces its processor speed to 1/2 = 0.5. When τ_1 preempts τ_3 at $t = 3$, as shown in Figure 5(c), it executes at its initial full speed.

Work-demand analysis

The work-demand analysis online calculates available slack time at the beginning of each job. The slack time includes both static and dynamic slack time to the deadline of this job. This job makes use of all available slack time to execute at a reduced processor speed. Figure 6 shows an example of 3 tasks all released at $t = 0$. Each task executes at the full-speed initially. When τ_1 begins at $t = 0$, it has $5 - (1 + 1 + 1) = 2$ static slack before its deadline at $t = 5$. τ_1 makes use of this slack to reduce its processor speed to 0.33, as shown in Figure 6(b). When τ_1 completes earlier at $t = 2$, τ_2 calculates that it has 1 dynamic slack before its deadline at $t = 6$. Accordingly, τ_2 reduces its processor speed to 0.5, as shown in Figure 6(c).

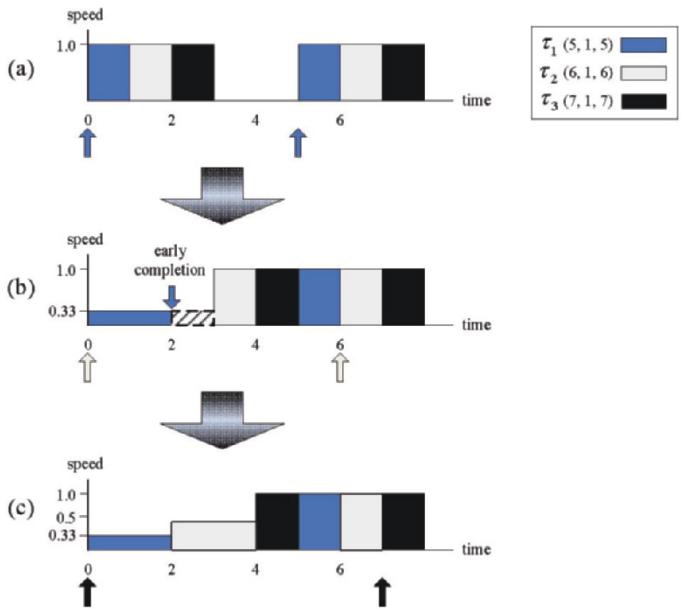


Figure 6. The work-demand analysis

Utilization update

The actual processor utilization during run-time is often lower than the worst-case processor utilization. The technique of utilization update estimates the required processor performance at the current scheduling instant by recalculating the expected worst-case processor utilization using the actual execution times of completed jobs. The executing processor speed is adjusted according to the updated processor utilization.

2.3 Real-Time DVS Scheduling algorithms

Among all related work, Yao et al. (1995) first presented an off-line optimal algorithm of $O(N^2)$ to schedule real-time tasks using DVS in a dynamic-priority system, where N denotes the number of tasks. The following introduction of DVS algorithms is divided into two parts: dynamic-priority DVS algorithms and static-priority DVS algorithms. Referred dynamic-priority algorithms include Shin et al. (2000), Pillai and Shin (2001), Kim et al. (2002), Aydin et al. (2004), and Lee and Shin (2004). Referred static-priority algorithms include Shin and Choi (1999), Shin et al. (2000), Krishna and Lee (2003), Pillai and Shin (2001), Quan and Hu (2001), Quan and Hu (2002), Kim et al. (2003), Yun and Kim (2003), Saewong et al. (2003), and Mochocki et al. (2005).

2.3.1 Dynamic-priority DVS algorithms

lppsEDF (Shin et al., 2000)

This algorithm, called lppsEDF, assumes a workload of periodic tasks and earliest-deadline-first (EDF) (Liu and Layland, 1973) real-time scheduling. This algorithm offline uses the method of minimum constant speed to determine a minimum processor speed equal to its total utilization and initially execute each task at this speed. When tasks run at their WCET, no processor idle time exists. This algorithm applies the technique of NTA stretching to utilize available dynamic slack. One restriction of lppsEDF is that dynamic slack calculation only takes place when there is only one job in the ready queue.

ccEDF (Pillai and Shin, 2001)

This paper proposed an efficient algorithm for calculating a minimum processor speed at completion of each job. The calculation is based on the method of utilization update. Because such calculation is only carried out at completion of a job, it may not fully utilize processor utilization. Furthermore, this approach does not adapt well to a dynamic workload where tasks carry a wide range of execution times.

To improve the performance of ccEDF, this paper proposed another aggressive algorithm, called LA-EDF. This algorithm first calculates the workload that must be done before the next deadline. It then sets the lowest possible speed to complete this minimum workload and defer as much workload as possible. The deferred workload is scheduled to be executed at a higher processor speed to avoid missing deadline. However, when a task completes earlier than its WCET, such a raise of speed may not be necessary. As a result, it adapts well to a dynamic workload and exhibits a better utilization of slack time.

A real-time DVS algorithm achieves better energy saving in a processor assuming a continuous range of speeds than one assuming a discrete range of speeds. However, in reality, a processor supports only a limited set of speeds. This paper (Rao et al., 2004) presented a method called Pseudo-Level Generating Algorithm (PLG) to be used in conjunction with any real-time DVS algorithm. This method partitions the execution of task into several slots and determines a processor speed for each slot. The result is stored in an array to be used in a table-lookup way during run-time. The method of PLG is integrated into LA-EDF to improve its energy saving by 25%.

lpSHE (Kim et al., 2002)

The approach of lpSHE also assumes a periodic workload and EDF scheduling. When a high-priority task completes earlier than its WCET, its available slack is allocated to low-priority tasks. The calculation of slack time is online and involves the techniques of minimum constant speed, NTA stretching, and priority-based slack stealing. The time complexity of this algorithm is $O(N)$ and its space requirement is marginal. The experimental results show that this algorithm reduces energy consumption by 20~40% over the lppsEDF algorithm.

AGR (Aydin et al., 2004)

This paper first proved that solving a real-time DVS problem is equivalent to solving a reward-based scheduling problem with concave reward functions. By transforming real-time DVS scheduling to reward-based scheduling, it proposed an optimal static solution for a real-time DVS scheduling problem where each task executes at their WCET. In addition, an online speed reduction algorithm called AGR is presented. Similar to LA-EDF, AGR chooses a lower processor speed first and later switches to a higher speed to complete deferred work. The amount of deferred work is customizable. The reduction in energy consumption is not directly related to the amount of deferred work. It requires careful analysis on distributions of actual execution times to determine deferred work for minimum energy consumption.

OLDVS (Lee and Shin, 2004)

OLDVS is an on-line real-time DVS algorithm that assumes no periodic workload or any priori information such as periods and arrival times of tasks. This algorithm first uses the method of minimum constant speed to determine an initial processor speed. At each completion of a task or preemption from a higher-priority task, it allocates available dynamic slack to the highest-priority task in the ready queue. The calculation of dynamic slack is based on the method of utilization update. Such calculation and allocation of dynamic slack is carried out in $O(1)$. The experimental results show that its performance is considerably better than previous real-time dynamic-priority DVS algorithms.

2.3.2 Fixed-priority DVS algorithms

lppsRM (Shin, et al., 2000)

This algorithm assumes a periodic workload that is initially scheduled by Rate-Monotonic scheduling (Liu and Layland, 1973). This algorithm consists of an off-line and an on-line component. The off-line component calculates available static slack and uses the method of minimum constant speed to set an initial processor speed. The on-line component uses the technique of priority-based stack stealing to determine available slack and reduce the processor speed accordingly.

Figure 7 shows an example of three tasks initially scheduled by RM. It first calculates a speed scaling factor, denoted by η_i, for each task τ_i. This process is illustrated below

$$\eta_1 = \min(C_1/P_1) = 0.5 ,$$

$$\eta_2 = \min\{(C_1+C_2)/P_1, (2C_1+C_2)/P_2 \} = 0.7 ,$$

$$\eta_3 = \min\{(C_1+C_2+C_3)/P_1 , (2C_1+C_2+C_3)/P_2 , (3C_1+2C_2+C_3)/P_3) = 0.67 ,$$

$$\eta = \max(\eta_1, \eta_2, \eta_3) = 0.7 .$$

Task	WCET	Period
τ_1	5 ms	10 ms
τ_2	2 ms	15 ms
τ_3	1 ms	30 ms

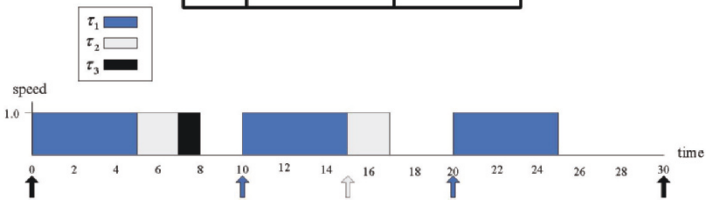


Figure 7. A task set

where C_i and P_i denotes the WCET and period of τ_i, respectively. Figure 8 shows the schedule and processor speeds of each task after obtaining all speed scaling factors. The process of on-line slack stealing only takes place when there is only one job in the ready queue.

ccRM (Pillai and Shin, 2001)

This algorithm is also based on a periodic workload executed by RM. It also consists of an off-line and an on-line component. The off-line component uses the method of minimum constant speed. The on-line component uses the method of NTA stretching. Because this algorithm speeds down the processor whenever there is slack time, without restriction on the number of ready jobs as demanded by lppsRM, it consumes less energy than lppsRM.

We use the same workload shown in Figure 7 to illustrate the process of ccRM. We first calculate a constant speed, denoted S_i, for each task τ_i. The initial speed, denoted by S, is next set to the maximum of all calculated speeds.

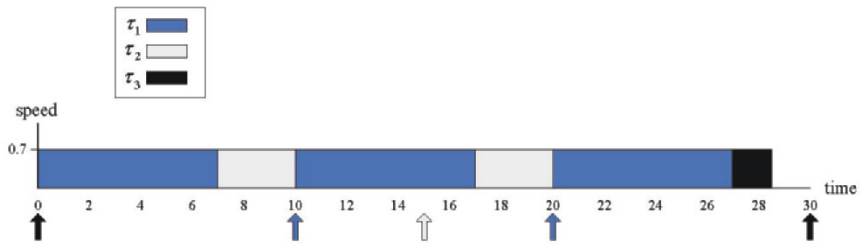


Figure 8. lppsRM

$$S_1 = \frac{\left\lceil P_1 / P_1 \right\rceil * C_1}{P_1} = 0.5,$$

$$S_2 = \frac{\left\lceil P_2 / P_1 \right\rceil * C_1 + \left\lceil P_2 / P_2 \right\rceil * C_2}{P_2} = 0.8,$$

$$S_3 = \frac{\left\lceil P_3 / P_1 \right\rceil * C_1 + \left\lceil P_3 / P_2 \right\rceil * C_2 + \left\lceil P_3 / P_3 \right\rceil * C_3}{P_3} = 0.67,$$

$$S = \max(S_1, S_2, S_3) = 0.8$$

We use $f(t)$ to denote the processor speed at time t and D to denote the first deadline after the current scheduling instant. This algorithm divides the total workload in the interval [t, D] by (D – t) to obtain $f(t)$.

$$t = 0, \quad \Rightarrow \quad D = 10, \quad S(0) = \frac{C_1 + C_2 + C_3}{P_1} = 0.8,$$

$$t = 10, \quad \Rightarrow \quad D = 20, \quad S(10) = \frac{C_1 + C_2}{P_1} = 0.7,$$

$$t = 20, \quad \Rightarrow \quad D = 30, \quad S(20) = \frac{C_1}{P_1} = 0.5.$$

(Krishna and Lee, 2003)

This paper proposed an on-line real-time DVS algorithm for a cyclic system where each task has the same period and deadline. Each task is assigned a fixed priority and is executed either at the lowest possible speed or at the highest possible speed. At completion of each job, it first calculates available slack time. It next speeds down the processor to the lowest possible speed to execute the next job for a longest period of time at the condition of keeping its deadline. They conducted a comprehensive experiment to the performance of the proposed algorithm. The experimental results show that, for a workload of a wide range of execution times, a processor with an infinite number of speed levels actually consumes more energy than a processor with only two speed levels. A similar result was also presented in Ishihara et al. (1988). They concluded that two levels of speeds are sufficient for most power-aware embedded systems.

lpWDA (Kim, et al., 2003)

Similar to previous work, this algorithm still focuses its discussion on a periodic workload executed by RM. It first proposes the technique of work-demand analysis for slack time estimation. When a job is ready to execute, this technique carefully examines every other job before its deadline to determine available slack. Such analysis obtains more slack time than previous work that includes less information in their estimation. Consequently, it delivers better energy saving by the utilization of more slack time. Its run-time complexity is at $O(n)$ and can be reduced to $O(1)$ in some cases.

Figure 9 uses the task set shown in Figure 7 to illustrate the operation of this algorithm. Let τ_i denote the job to be executed at the current instant t, d_i denote its deadline, and w_i denote its remaining workload. Let $H_i(t)$ and $L_i(t)$ denote the workload of higher-priority and lower-priority jobs in $[t, d_i]$, respectively. Again, we use $f(t)$ to denote the processor speed at time t. The calculation of processor speeds by lpWDA is shown below.

$$t = 0, \quad \Rightarrow \quad S(0) = \frac{w_1}{d_1 - t - H_1(0) - L_1(0)} = 0.625,$$

$$t = 8, \quad \Rightarrow \quad S(8) = \frac{w_2}{d_2 - t - H_2(8) - L_2(8)} = 1.0,$$

$$t = 10, \quad \Rightarrow \quad S(10) = \frac{w_1}{d_1 - t - H_1(10) - L_1(10)} = 0.5,$$

$$t = 20, \quad \Rightarrow \quad S(20) = \frac{w_1}{d_1 - t - H_1(20) - L_1(20)} = 0.71,$$

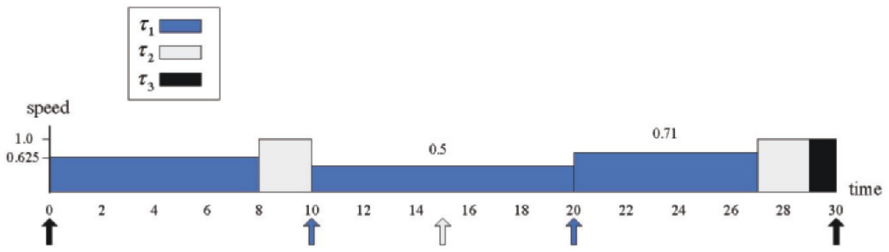


Figure 9. lpWDA

PMclock (Saewong et al., 2003)

This algorithm adopts a periodic workload executed by RM. It uses the technique of priority monotonic to make use of slack time. This algorithm intends to use a faster processor speed for a higher-priority task. For each task τ_i, it first statically determines the amount of static slack claimed from any higher-priority task in a critical interval. The slack time is used to speed down the processor for τ_i. However, if such a calculation results in a faster speed for a lower-priority task than a higher-priority task, the former will be executed at the same speed as the latter to comply with its priority-monotonic requirement. The available dynamic slack is simply allocated to the next lower-priority job to be executed. For the example shown in Figure 7, this algorithm sets $S_1 = S_2 = 0.7$ and $S_3 = 0.67$. Since there is dynamic slack between [28, 30], τ_3 makes use of it to speed down S_3 to 0.3.

VSLP (Quan and Hu, 2001)

This algorithm is based on a non-periodic workload where each task has a fixed priority. It utilizes the technique of essential interval to speed down a processor for energy saving. A 2-step iterative process is deployed to determine a processor speed for each interval. First, an essential interval for each job is located. Among all intervals, the one with the largest utilization is scheduled to be executed at a speed equal to its utilization. Let this interval belong to a job J_i. Secondly, J_i and every higher-priority job released in this interval are removed from the workload. For each of the rest jobs overlapped with this interval, either its deadline is shifted to the beginning of the interval or its release time is deferred to the end of this interval. Finally, this interval is removed from the schedule and the iterative process continues to locate the next essential interval.

OPT_FP (Quan and Hu, 2002)

This algorithm adopts a non-periodic workload where each task has a fixed priority. It uses the technique of EDF transformation to utilize slack time for speed reduction. This algorithm first presents a heuristic approach to transform a set of fixed-priority jobs into an equivalent set of EDF-based jobs by adjusting their deadlines. However, the same execution order is maintained in the new set. The new set of jobs is next executed by an optimal low-power EDF scheduling algorithm. This paper proved that an optimal low-power EDF schedule of the EDF-transformed set is the same as an optimal DVS schedule of the original set. Finally, the time complexity of EDF transformation is $O(N!)$.

FP_TAS (Yun and Kim, 2003)

This algorithm adopts a non-periodic fixed-priority workload. The contributions of this paper are two folds. First, it proves that the problem of optimal real-time DVS for a fixed-priority workload is NP-hard. Secondly, it uses dynamic-programming formulation to reduce the time complexity of EDF transformation to $O(N^3)$, compared to $O(N!)$ required by the OPT_FP algorithm.

3. LOW-POWER DEVICE SCHEDULING

Dynamic Power Management (DPM) has been widely used in both commercial embedded systems and research work to reduce the power consumption of the I/O subsystem. In this section, we first describe several classical DPM policies. We next present DPM policies for real-time systems. The discussion continues to include energy-ware device scheduling and hybrid power management. Finally, we describe a couple of industry-design standards.

3.1 Classical DPM Policies

The basic idea of DPM is to switch idle devices to a low power state if the energy saving of this decision can compensate state transition overhead. The minimum length of an idle interval to save power is called the break-even time, denoted by T_{be} (Lu and Micheli, 2001). It is a characteristic of a device and independent of its workload. The accuracy of prediction on idle length becomes an important factor since a device can be put to a low power state if its idle period is longer than T_{be}. Figure 10 shows an example where a task τ_i issues an I/O request on a device k_x in [t_0, t_1] and [t_2, t_3]. We assume that the idle interval [t_1, t_2] is longer than k_x's break-even time. k_x is put in to

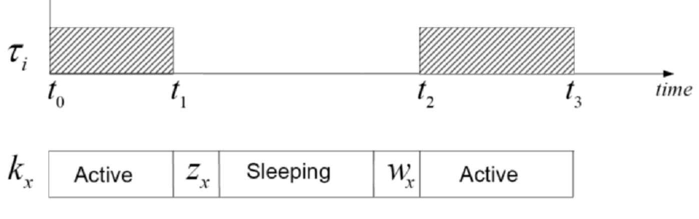


Figure 10. The concept of dynamic power management

sleep mode for energy saving during [t_1, t_2]. Let z_x and w_x denote the shutdown and wakeup latency of k_x. The example shows that, in order to save energy and avoid delay of execution by an unready device, the accuracy of idle-time prediction is important for a DPM policy.

Existing DPM policies are classified into three categories: timeout-based, predictive, and stochastic. A timeout-based policy is widely used in embedded systems due to its simplicity. It puts a device in a low-power state when its idle interval is longer than a predefined threshold. The main drawbacks are that the device wastes energy during the idle interval and the device may not remain in idle for at least T_{be}. Karlin et al. (1994) proposed to use T_{be} as a timeout interval and showed that this choice leads to an energy consumption at worst twice the energy consumed by an ideal policy.

Predictive policies are investigated to improve the performance of timeout-based policies. If an idle period of a device is predicted to be longer than its break-even time, the device switches to a low-power state right after it becomes idle. Such a policy uses past information to predict the length of the next idle period. The L-shape policy (Srivastava and Chandrakasan, 1996), the adaptive learning tree (Chung et al., 1999) and the exponential-average policy (Hwang and Wu, 1997) are three classical predictive algorithms. S. Irani et al. (2003) presented a deterministic online DPM policy on multi-state devices and proved that its performance is 2-competitive to an optimal algorithm. Its experimental results show that this algorithm delivers the best performance, in comparison to other known predictive DPM algorithms.

Timeout-based and predictive policies are often formulated heuristically. Stochastic policies (Benini et al., 1999) model the arrival times of requests and device power-state changes as a stochastic process, such as a Markov process. This work solves the problem of finding optimal tradeoff between performance and power as a stochastic optimization problem. Finally, two excellent surveys and performance comparison for these DPM policies can be found in Lu and Micheli (2001) and Benini et al. (2000).

Lu and Micheli (2001) used the technique of filter driver to implement and evaluate a number of DPM policies. A filter driver is inserted between the operating system kernel and a low-level device driver. It intercepts communications between these two software layers and issues switching decisions on behalf of a DPM policy. They proposed six criteria to evaluate a DPM policy: power, number of shutdowns, shutdown accuracy, interactive performance, and memory requirements. The experimental results show that no existing DPM policy achieves an A grade in all columns. Particularly, a policy with the best power efficiency performance results in low interactive performance and requires more memory. A designer can make use of such information to select an appropriate DPM policy for their specific hardware and software requirement.

3.2 DPM Policies for Real-Time Systems

Many embedded systems are designed to meet real-time constraints, such as automobile, avionics, medical applications, multimedia, defense applications, and telecommunication. These systems are required to meet both functional and timing requirements. Classical DPM policies cannot directly be applied to these systems because of their non-deterministic nature. A real-time DPM policy must carefully consider each power characteristic of a device, such as its break-even time, wake-up latency, and power consumption at different operational modes. Switching off devices at a wrong time can potentially result in a task missing its deadline.

3.2.1 Hard real-time DPM polices

For hard real-time system, one straightforward approach of DPM is to statically construct a huge switching-decision table for each device in a hyper period according to each task's worst-case execution time (WCET) and static I/O access-pattern. We use this table during runtime to switch on or off a device at each scheduling instant. Swaminathan et al. (2003) proposed a similar static approach, called LEDES, for a tick-driven scheduling system that assumes each task has a fixed execution time. LEDES considers each slice of a job as an independent workload with the identical device-usage list. LEDES also assumes that each device’s power-mode transition latency is less than or equal to the execution time of the shortest job. By making this assumption, LEDES safely determines a switching decision by looking ahead only one job without considering the device's break-even time. LEDES may consume more energy when an idle period of a device is less than its break-even time. In addition, it is applicable only for a deterministic system equipped with a large amount of memory. This paper also presented another algorithm called MUSCLES to handle devices with more than two power states. MUSCLES estimates the number of scheduling instants before a device is accessed and switches it on at the latest scheduling instant without missing a deadline. Both approaches rely on static information such as a constant processor speed and a fixed execution time. Consequently, they cannot make use of dynamic run-time information such as an adjustable processor speed, variable execution time, and a configurable task set.

Liu and Chou (2004) found out that the break-even analysis is crucial for energy saving only when a device has two power states. However, many modern electronic devices support multiple power states and, thus, the break-even analysis only achieves sub-optimal solution. This paper presented a new approach to calculating an optimal switching sequence of each device for a given deterministic task set. The result is stored in a lookup table to be retrieved during run-time for Θ(1) lookup performance.

3.2.2 Soft real-time DPM policies

For soft real-time systems, several commercial RTOS vendors have already supported DPM in their products. IBM and Montavista proposed an architecture supporting aggressive DPM for embedded systems (Brock and Rajamani, 2003). Developers can assign different weights to tasks, such that RTOS executes a task at a power/performance level matching its assigned weight. For example, a multimedia playback application requiring real-time performance can demand a high-power high-performance weight while other non-real-time tasks execute at a low-power state. This module is maintained as an open-source project at http://dynamicpower.sourceforge.net/. It provides Linux 2.6 kernel patches and setup scripts for a list of supported platforms. The code and documentation found at this URL address may be useful for gaining the latest information and future directions for DPM support in Linux kernel.

QNX proposed an application-driven model that enables fine-grained power consumption control of each I/O subsystem (Ethier, 2003). A user-mode power manager issues power-state changes requested by applications. When an application requests the power manager to switch off a device, this request will be filtered by the power manager to make sure it will not block another application accessing the same device. This mechanism allows a designer to create a customized power management policy without modification on the level of kernel code or device drivers.

3.3 Energy-Aware Device Scheduling

A DPM policy saves energy by putting idle devices to a low-power state. By putting together tasks that access the same devices, we can create more switch-off opportunities and achieve better energy saving. We present here several approaches of reordering the execution sequence such that idle periods are grouped instead of scattered.

3.3.1 Soft real-time device scheduling

Lu et al. (2000) proposed an on-line low-power device scheduling for non-real-time systems. At each scheduling instant, it first selects a task whose device usage list is the same as the previous task. If such a task cannot be found, it next finds a task that creates a switch-off opportunity. When both steps fail, it selects a task with the best potential of providing switch-off opportunity in the future. The experimental results show that this approach saves up to 33% energy and reduces around 40% power-state changes.

Weiseel et al. (2002) presented Coop-I/O, a power-management interface specially designed for energy-aware applications. Through Coop-IO, an application can declare open, read, and write operations as deferrable or abortable. An operating system makes use of this information to delay and cluster I/O requests in order to reduce the number of power-state changes and keep a device in a low-power state for as much as possible. Coop-IO was implemented into the IDE disk driver and Ext2 file system of Linux kernel. Several practical scenarios are presented to utilize this new I/O interface. The experimental results show that this mechanism can save energy by up to 50%.

The major difference between Weiseel's (2002) and Lu's (2000) method is that Coop-I/O enables an application to pass the delay time of an I/O request when it is issued; programmers require no global knowledge of all I/O requests for their programs. On the other hand, Lu's approach arranges the execution order of tasks to create energy-efficient device-access pattern, while Coop-I/O schedules device requests directly without involving the task scheduler.

Cai and Lu (2005) presented a method to combine memory and disk power management for achieving better energy saving. They identified a tradeoff between memory and disk energy consumption. The disk can spin down longer to save energy at the increase of memory size. However, such an energy saving may not compensate the power consumption by the additional memory. In this paper, they proposed an algorithm to predict the number and inter-arrival time of disk I/O under different memory size. This information becomes necessary for a power manager to determine the size of memory and the timeout interval for shutting down a hard disk.

3.3.2 Real-time device scheduling

Device scheduling in a real-time system requires sophisticated analysis to consume minimum energy without violating any timing constraint. Figure 12 shows an example to illustrate the complexity of this problem. There are two periodic tasks T_1 and T_2. T_1 requests device 1 and T_2 requests device 2. T_1's priority is higher that T_2. Figure 11 shows an inefficient way of accessing both devices while Figure 12 clusters requests of a same device. Obviously, Figure 12 exists more opportunities to switch off idle devices.

Finding a feasible low-power device schedule for a real-time task set that consumes minimum energy had been proved NP-complete. The main idea of proof is to reduce this problem to the problems of sequencing within

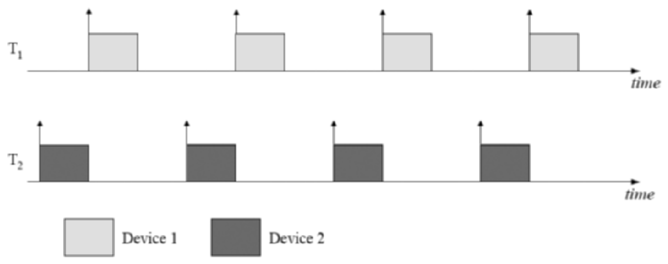


Figure 11. The execution of two periodic tasks

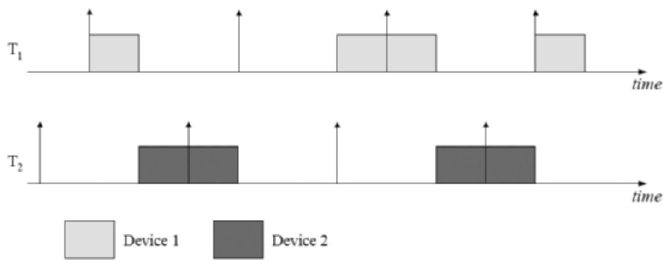


Figure 12. Energy-efficient device scheduling

intervals (Swaminathan and Chakrabarty, 2005). Swaminathan and Chakrabarty (2002) proposed an algorithm called EDS to find an optimal solution for this NP-complete problem. They use a pruning technique to generate a schedule tree and iteratively prune branches in which the optimal solution does not exist. The schedule tree is pruned based on two factors – time and energy. Temporal pruning is performed when a partial schedule of jobs causes missed deadlines. Energy pruning is performed when a partial schedule induces higher energy consumption. Eventually, the pruning process leads to a leaf node with least energy. The energy-optimal device schedule can be obtained by backward tracing the path from the leaf node to the root node. However, the proposed method requires a lot of memory and computation time for a large-scale system. It is only suitable for small embedded systems.

Due to the high complexity of EDS, Tian and Arslan (2003) proposed a Genetic-Based algorithm to generate a near-optimal device schedule for a set of real-time tasks. When compared with other algorithms, the genetic algorithm requires less memory and computation time and is more suitable for a large-scale system.

SURE (Slack Utilization for Reduces Energy) is an on-line real-time algorithm for a dynamic-priority system (Krishnapura et al., 2004). If a device is switched on, ready-queue jobs accessing this device are executed first. Alternatively, if a device is in sleeping mode, jobs that access this device will be delayed for as long as possible at the condition of meeting its deadline. This method reduces the number of power-state changes and keeps a device in the idle state for a longest period of time. The SURE algorithm can be executed offline to generate a cyclic schedule for online execution. It can also be executed online to adapt to a dynamic workload and achieve better energy saving. The major side-effect of EDS, Genetic-Based, and SURE is that jobs of the same task may execute one after the other. Such an arrangement maximizes the activation jitter of a task. In certain real-time control systems that demand smooth playback, this is not a desirable feature.

3.4 Hybrid Power Management Technique

The current leakage in standby mode is increasing with the advances of CMOS technology and must also be taken into account. The technique of dynamic voltage scaling (DVS), although reducing the processor energy consumption, extends the execution time of a task and increases the energy consumption of the I/O subsystem. There exists a trade-off between DVS and DPM scheme. A combined approach is needed to address the issue of reducing system-wide energy consumption.

3.4.1 Hybrid scheduling for real-time systems

Kim and Ha (2001) proposed the first approach that integrates the techniques of DVS and DPM for real-time systems. It partitions the execution of a task into a sequence of time slots and switches off idle devices on a slot-by-slot basis. They identified that there is a significant trade-off between DPM and DVS under different scheduling conditions. The main idea is to switch off a device if the time to the beginning of the next period is greater than its break-even time. For simplicity, the proposed method ignores device-transition latency and leaves a device in sleeping mode until needed. It also assumes that the power consumption in standby mode (i.e., a device is on but not serving requests) is the same as that in active mode (i.e., a device is on and serving requests).

Jejurikar and Gupta (2004) developed an off-line algorithm to consider both processor energy leakage and standby energy consumption of devices in determining a processor speed of each task. This algorithm first computes the critical speed for each task. If it is infeasible to schedule this task set at this speed, it next increases the processor speed to achieve feasibility. An iterative heuristic method is used to select a task and its processor speed. However, this approach completely ignores the impact of DVS on device power management.

Chu et al. (2005) present COLORS, a composite low-power real-time scheduling algorithm that applies DVS on top of a novel real-time DPM policy. COLORS is an on-line algorithm that adapts well to dynamic workloads resulted from variable execution times. It utilizes slack time and DVS to maximize opportunities for switching off idle devices. In addition, the development of COLORS takes a practical approach to consider every parameter of a device. A couple of matrices are identified to primarily determine the performance of real-time DPM. When compared with a theoretical optimized version which assumes knowledge of actual workloads, COLORS still delivers comparable performance.

3.4.2 DVS during I/O

In order to explore the impact of DVS on device activities, Acquaviva et al. (2001) described a software-controlled approach to adaptively minimize energy consumption in real-time multimedia embedded systems. This approach optimizes energy consumption by dynamically adjusting the processor speed to the frame rate requirements of incoming multimedia streams. It uses offline application profiling to obtain a performance-frequency mapping that is used to calculate an optimal speed for energy saving. In contrast, Choi et. al (2004) adopted a monitoring unit to perform workload decomposition. The profiling report obtained by the monitoring unit reveals how and when the CPU is stalled during the execution of each application. Based on this information, we can determine a suitable processor speed for each task to reduce energy consumption while minimizing the impact on its runtime performance.

3.5 Industry Design Standard

In 1997, ACPI (Advanced Configuration and Power Interface) was proposed by a number of major industry players, including Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba, as an open industry specification (Hewlett-Packard, et al., 1997). ACPI establishes industry-standard interfaces for power management on laptops, desktops, and servers. This set of defined interfaces enables new power management technology to evolve independently in system libraries, operating systems, and hardware. Figure 13 lays out the software and hardware components relevant to ACPI and their architecture. This specification describes the interfaces between each component. With the availability of ACPI, several commercial operating

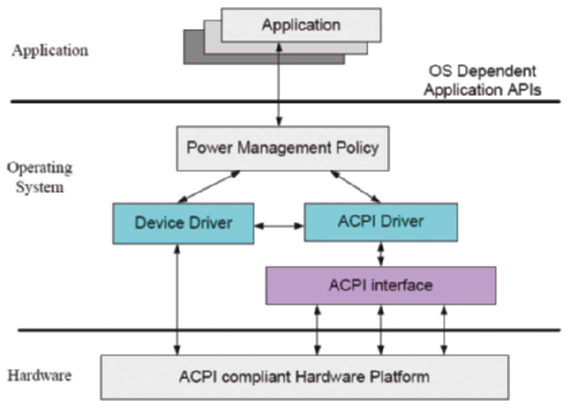


Figure 13. ACPI architecture

systems start to implement ACPI-compliant device power management. Examples include, but are not limited to, Microsoft OnNow (Microsoft, 2001) and ACPI4Linux (SourceForge, 2002) projects. However, due to its complexity, it is hard to implement the whole set of ACPI interfaces on small embedded systems. The latest information of ACPI can be found at http://www.acpi.info/.

Anand et al. (2004) provided a new set of interfaces that allows application-level information to be used in a DPM policy. Such interfaces are not available in the ACPI standard. These interfaces allow an application to not only query power information of I/O devices but also provide information on application behaviors for better power management. For example, when a disk is in standby mode, an adaptive application would rather fetch a small file from a networked machine if such an operation incurs less energy. On the other hand, for a large file, an application should retrieve from a local disk since reading from a networked machine will demand more energy. The availability of these interfaces enables an application to issue an appropriate DPM decision after carefully examining its own execution behavior.

4. DEVICE DRIVER DEVELOPMENT

An embedded system is generally equipped with various peripheral devices, and handles external events through input/output (I/O) channels. I/O jobs constitute a large proportion of tasks processed by an embedded system. Therefore, software drivers controlling I/O devices play a very important role in the embedded software design, and also significantly impact the efficiency, performance and real-time properties of an embedded system.

This section presents the characteristics, operations and design issues of a device driver for an embedded system.

4.1 Characteristics and Operations of Embedded Device Drivers

Embedded software markedly differs from general-purpose software. It works tightly with embedded hardware, and handles interactive and real-time events. A device driver controlling a specific hardware device cooperates with an embedded OS (EOS), enabling embedded applications to process external events quickly and efficiently. The interrupt-handling procedures of an EOS and the device drive designs significantly affect the efficiency, the functional and real-time correctness of an embedded system. Unlike a general-purpose OS focusing on flexibility, portability, configurability and layered structures, an embedded device driver and its related EOS functions, such as interrupt handlers and I/O subsystems, aim to improve the efficiency and effectiveness of the execution, code size and real-time characteristics. Figure 14 shows an example of the I/O procedures of a general-purpose OS. During the driver initiation phase, a device driver must first associate its interfaces, such as `dev_open()`, `dev_read()`, `dev_write()` and `dev_close()`, to the generic interfaces of a standard I/O subsystem of the OS, such as `open()`, `read()`, `write()`, `close()`. The procedure maps the I/O

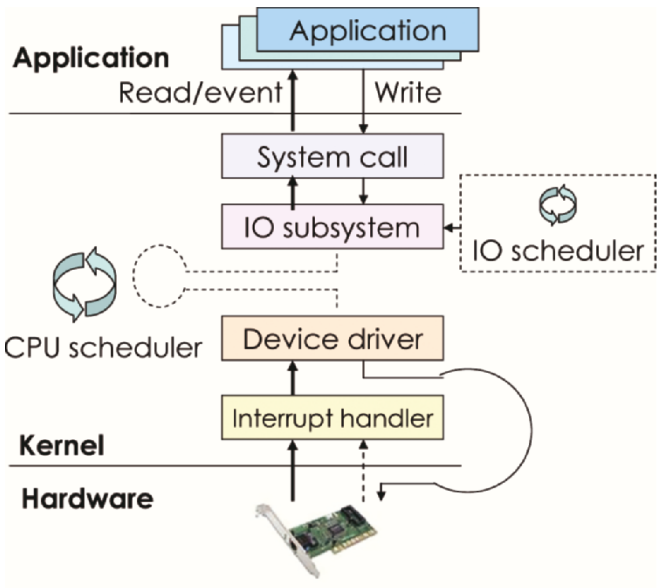


Figure 14. I/O procedures of a general-purpose OS

subsystem to the specific driver and device. Additionally, the device driver must attach its interrupt service routine (ISR) to the interrupt handler, which is defined by the OS. The ISR can then be invoked to process the external event triggered by a particular device. Once an application requests I/O operations such as `write()` or `read ()` through the system call interface, these requests are forwarded to the I/O subsystem. Depending on the OS implementations, the I/O subsystem might have an internal I/O scheduler to merge or shuffle I/O requests in order to improve the I/O efficiency. After the application generates an I/O request, the application is set to idle, and waits the I/O response if the I/O request can not be finished immediately. The scheduler is then called to pick up another task in the ready queue to run. If an interruption occurs, the CPU (Central Processing Unit) is forced to stop its current execution, and jumps to the ISR to process the event. This interruption event might change the states of tasks. For instance, if a read request is finished and an interruption occurs, then the task waiting for the read response is moved from the waiting queue to the ready queue and can be scheduled again. Notably, the design aims of the interruption-handling mechanism, device driver, and I/O subsystem of a general-purpose OS focus mainly on the structured and the standardized interfaces to facilitate the development and operations of the device drivers. The improvements of efficiency and real-time characteristics of a device driver for embedded software and embedded OSs are described next.

Typically, the embedded software of an embedded system can be implemented by two approaches, non-OS-based and EOS-based. Non-OS-based implementation implies that no OS is involved in embedded software. The programmers must write a control program, device drivers and other supporting routines to perform the embedded software functions. Non-OS-based implementation is more efficient than EOS-based implementations in terms of code size and execution speed, but requires programmers to handle every detail of the embedded software. Non-OS-based implementation is generally applied to simple embedded systems. Conversely, embedded software based on the EOS-based implementation relies on the services offered by an EOS. Although EOS-based embedded software requires extra execution memory space, needs more flash memory to store the program image, and involves EOS overhead, it significantly reduces the development time and the complexity of the embedded software. The EOS-based approach is suitable for complex embedded systems, where it speeds up the development process. Figures 15 and 16 show the software architectures of non-OS-based and EOS-based embedded software, respectively, and also illustrate the interrupt timing diagrams.

Figure 15 shows an example of I/O procedures for a non-OS-based embedded software system. An embedded application, which is a control program, accesses peripheral devices via device drivers. A device driver

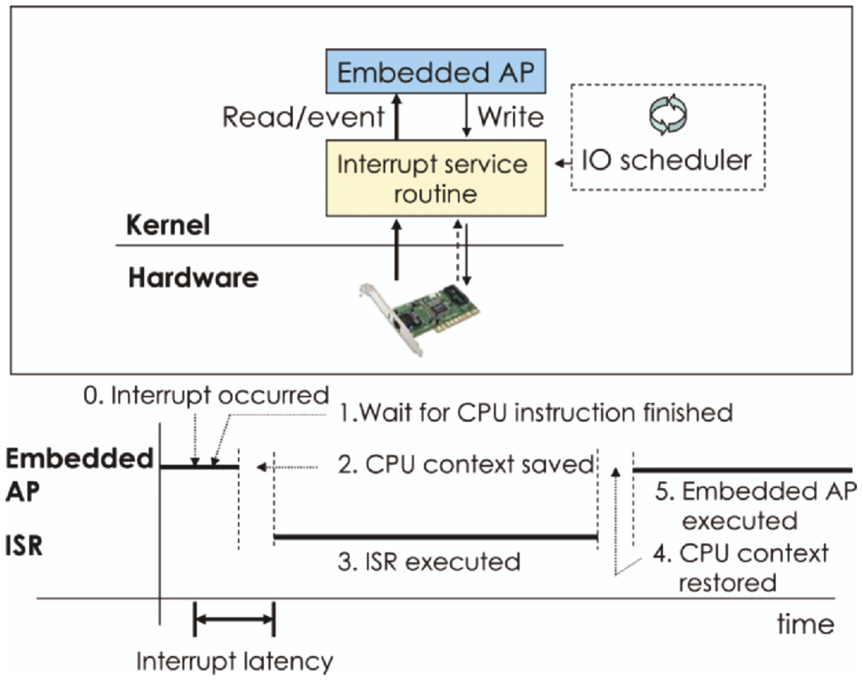


Figure 15. I/O procedures and interrupt timing diagram based on a non-OS implementation

implements an interrupt service routine (ISR) that handles the interrupts from the device. The ISR can implement I/O schedulers inside to further merge or re-schedule the I/O requests for a better performance. The procedures to process an I/O request once an interrupt arises are:

1. The CPU must finish the instruction that is currently executed, and stops executing the current embedded application.
2. The CPU pushes the CPU contexts, such as the current program counter and the stack point, to the stack. The extra CPU contexts such as registers might also be automatically saved, depending on the embedded processor design. If the CPU does not push certain contexts that might be used during ISR to a stack, the ISR itself must save these CPU contexts. Furthermore, to prevent incoming interruptions confusing CPU states, the hardware disables all interruptions when an interruption occurs.
3. Before the ISR is executed, the CPU must lookup the ISR address stored in an interruption vendor table. This table-lookup task is performed by either hardware or software, depending on the CPU design. After the CPU obtains the ISR entry address, the CPU starts to run the ISR. The time between the interruption and the execution of the first instruction of ISR is defined as interruption latency. For a real-time embedded system, the interruption latency should be determined. The ISR might soon

enable interruptions again to allow incoming interruptions, to prevent the loss of interruptions. However, ISRs are generally non-reentry, so the interruption currently being processed by the ISR is disabled.

4. The ISR is finished, and the CPU context is restored.
5. The embedded application is then resumed.

The other implementation approach of the embedded software uses an EOS. Figure 16 shows an example of I/O procedures for an EOS-based embedded software. Entities denoted by blocks with dotted lines indicate that the entities might not exist for all EOSs, depending on the EOS implementations. An EOS might remove the wrapper layers and I/O subsystem layers within the kernel to improve the efficiency of I/Os processing. For instance, the drivers and programs in TinyOS are defined as components. Three possible component types exist in TinyOS, i.e. hardware abstraction, synthetic hardware and high-level software. Each component exports its own commands and then components are tightly integrated together. The I/O subsystem and wrapper functions are eliminated in TinyOS (Hill et al., 2000). Unlike the non-OS approach, the embedded AP running on top of EOS is an OS task. The task makes system calls, and the I/O requests are then passed to the I/O subsystem or directly mapped to the specific driver. A general-purpose OS might separate the interrupt service routine into two parts, i.e. top half and bottom half. The top half is a non-reentry, fast and small piece of code handling critical hardware actions. The bottom half, or so-called the deferred work, which is executed after the top half, supports program reentry, and can spend more time than the top half to process the rest part of the request. Not all EOS separates ISRs into two halves. The procedures handling the I/O request once an interrupt arises are:

1. As in the non-OS approach, a CPU finishes the instruction that is currently being executed, and stop running the current task.
2. The CPU pushes the current program counter, stack points or other related CPU contexts to a stack, and disables all incoming interruptions. The CPU then looks up the entry address of the ISR stored in an interruption vendor table.
3. Before executing the ISR, a kernel ISR entry function must be invoked to notify the kernel that an ISR is in progress. The kernel can then track the interruption nesting.
4. The CPU jumps to the entry address of the ISR, and the ISR starts to execute. Some EOSs separate the ISR execution into two halves, especially while the ISR needs more time to process an I/O request. To avoid blocking interruptions for a long period, the bottom half ISR is set to interruptable by all interruptions. No clear line exists between the top and bottom halves. Generally, the top half ISR must perform very fast, and can not be interrupted by the same interruption. Conversely, the

bottom half generally spends more CPU time but it is interruptable by the same external events.

5. To ensure that the interruption processing can be finished as soon as possible, the bottom half is invoked immediately after the top half is completed. Unlike a top-half ISR, the same interruption is enabled while executing the bottom half.
6. For a preemptive EOS, an interruption can preempt the current executed task. Therefore, after the ISR is finished, the scheduler is then called to pick up the most appropriate task to execute.
7. If the interruption picks up the original execution task, the CPU returns to it. Otherwise, a context switch to another task is required. For a non-preemptive EOS, the CPU must return to the original task after the interrupt is finished. The scheduler executes and picks up a new task to run after the original task releases the CPU or the allocated time slice expires. In this case, the embedded application suffers from a longer delay in responding to the interruption event than that in a preemptive EOS.
8. The original task is then resumed if it is picked up by the scheduler. Otherwise, another task is executed.

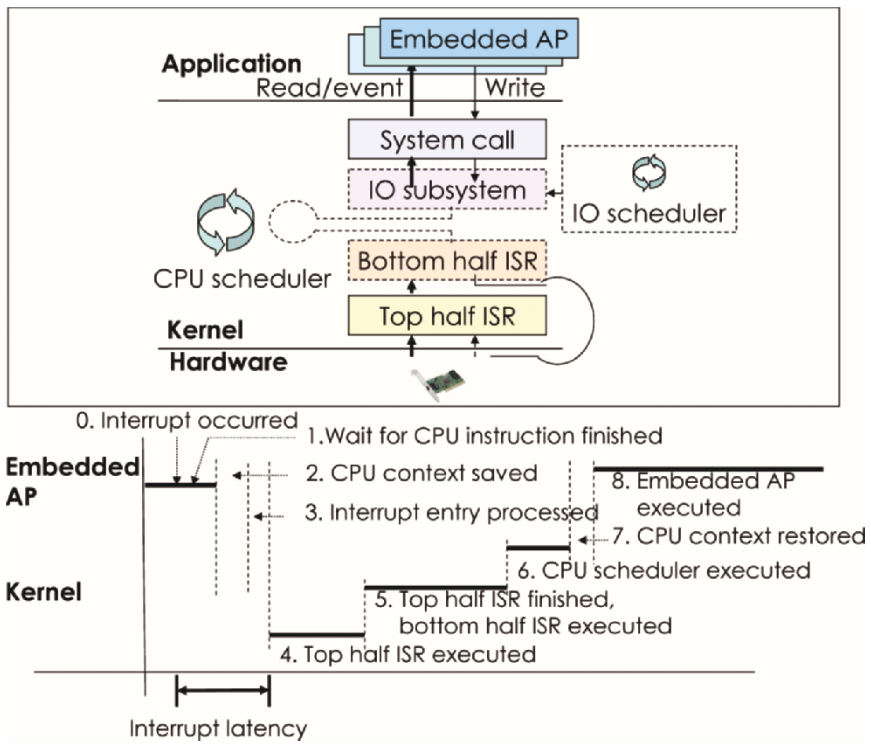


Figure 16. I/O procedures and interrupt timing diagram based on an EOS-based implementation

An embedded device driver and its related EOS kernel functions mainly concern the accurate control and real-time properties of the interrupt-handling procedures and the efficiency of the interrupt service routines.

4.2 Device Driver and Hardware/Software (HW/SW) Co-Design

Peripheral devices of an embedded system are often customized, and the hardware and software must be specifically designed. Conventional approaches to develop software generally can not proceed until the hardware is ready and fully tested. Such separated development methodologies lead to a long development time. Hardware/software (HW/SW) co-design technique can be applied to speed up the design and implementation process of hardware and software of an embedded device (Jerraya et al., 2003; Wang et al., 2003). Figure 17 shows an example of the embedded device design according to the co-design technique. The design process can be partitioned into three phases: the high-level design phase, the low-level design phase, and the implementation, integration and testing phase. The system designers first describe the embedded peripheral device from a high-level requirement perspective. Languages such as UML and SpecC can be used for high-level specification (Honda and Takada, 2003). The high-level design must describe the functionalities, features, timing, interactions and interfaces between the hardware and software of the device based on the requirements.

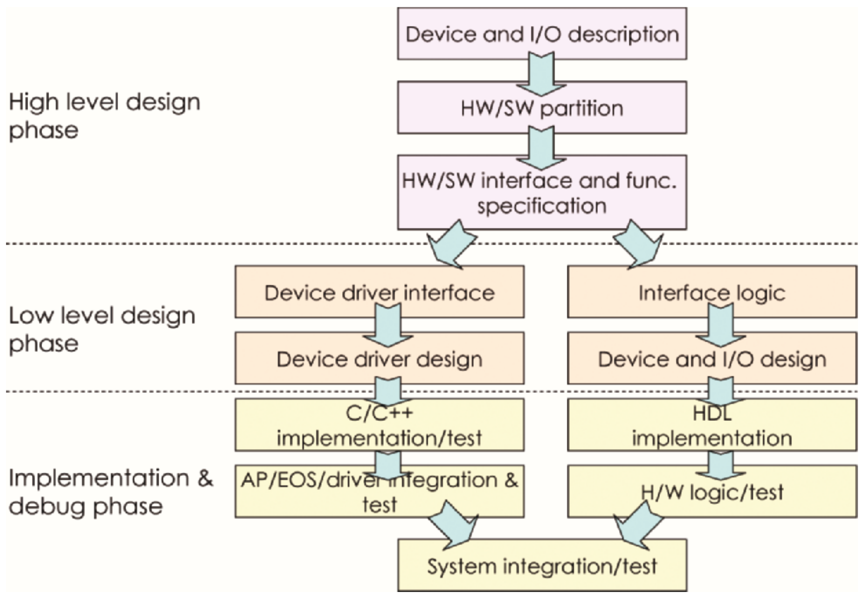


Figure 17. HW/SW co-design flow of an embedded device and its driver

No clear boundary exists between the hardware and software of an embedded device. A function block may be implemented on hardware to satisfy the requirements of the execution constraints, or may be implemented on software to fully use the computation power of an embedded processor, thereby reducing the hardware cost. The partition of HW/SW, particularly for a complex embedded system, needs a comprehensive and systematic analysis and investigation. After confirming the functional partitions of HW/SW, the interfaces between HW/SW can also be specified. The system level description based on UML or SpecC provides a high-level description of the hardware, software and their interfaces. The hardware and software design of the embedded device beyond the high-level design can be separated. The hardware low-level design according to the high-level description can be divided into the interface logic design and the I/O and device hardware design. The interface logic is a glue logic that maps the high-level interface descriptions to the precise hardware logic of the devices. Meanwhile, the device driver can be designed according to the high-level description of software. First, the device driver interfaces must be specified based on both HW/SW interfaces, and must also refer to the EOS interruption and interruption-handling mechanisms. The interface to the hardware is employed to control the hardware, and the interface to EOS is adopted to realize the device drive and interrupt service routines. Some reconfigurable embedded OSs, such as eCos, generailize the interfaces between hardware and software (Massa, 2002). For instance, the hardware abstraction layer (HAL) that lies between hardware and software in eCos provides an abstraction view, aiding the software development and providing re-configurability and flexibility. A uniform device driver (UDI) provides a reference model for the standardized interfaces between an OS and a device driver, and benefits the design and portability of the driver (Barned and Richards, 2002). However, current EOSs do not implement UDI but have their own mechanisms and interfaces to manage drivers and interruptions.

The implementation can proceed after the low-level design process is completed. Devices for general-purpose system could apply automatic code generation based on high-level and low-level designs. However, to improve performance and efficiency, manual implementation to optimize the codes is encouraged for embedded devices. Hardware and software of the embedded device testing is another critical task. The testing of embedded devices can be split into several stages. First, each HW/SW functional block is tested. Integrated hardware devices and software drivers are then verified separately. Finally, the HW/SW are integrated and tested. Hardware testing is fairly straightforward, and can be performed using various well-established hardware verification and validation methodologies. Conventional software engineering methodologies on software verification and validation, such as block box testing and white box testing, do not work well on embedded device drivers. Device drivers that work on kernel level and are tightly coupled with hardware are difficult to test offline. Testing over the target takes a long time but provides accurate test results. Device drivers can be tested by hardware simulation, but the simulations detail to timing behaviors is not easy to develop. The verification and validation technologies of embedded device driver, particularly hardware-software co-simulation, must be further studied.

The co-design method can significantly cut the development and debugging time of the embedded device, but has a number of challenges. First, the system methodologies to determine the HW/SW partition need to be established. Second, the verification and validation of embedded devices, particularly for hardware-software co-simulation, is another important research topic.

4.3 Improving the Efficiency and Interruption Latency of a Device Driver

An embedded system that handles frequent I/O events must consider the driver efficiency and interruption latency. The deterministic interruption latency is particularly important for a real-time embedded system. The interruption latency is fairly easy to determine in the non-OS approach (Weinberg, 2004), since software architecture consisting only of an embedded control program and interruption service routines is simple compared with the EOS-based implementation. When an interruption occurs, the CPU first completes the current instruction, pushes CPU contexts to a stack and then jumps to the specific ISR. For instance, the worse case fast interrupt (FIQ) latency in ARM6 is determined from: ① the time for the FIQ to pass through the CPU, which takes 3 clocks; ② the time to wait the CPU finishing the longest CPU instruction, which takes 20 clocks, and ③ the time for higher priority task and FIQ entry sequence, which takes 5 clocks. The time spent on these procedures has worst cases, and the interruption latency is determined. The internal interruption handler and scheduler design of the EOS-based approach influence the interruption delay suffered by the control program running in the application mode. The schedule hierarchy of an EOS design must be understood to identify the interrupt schedule behaviors and obtain the worse case interruption latency (Regehr et al., 2003). Figure 18 shows the schedule hierarchy of an embedded Linux kernel. The exception has the highest priority, and the hardware interruptions have the 2nd priority. Embedded Linux splits ISRs into top halves and bottom halves, so the top half of the ISR, i.e. `irq_action()`, is performed first. The interruption

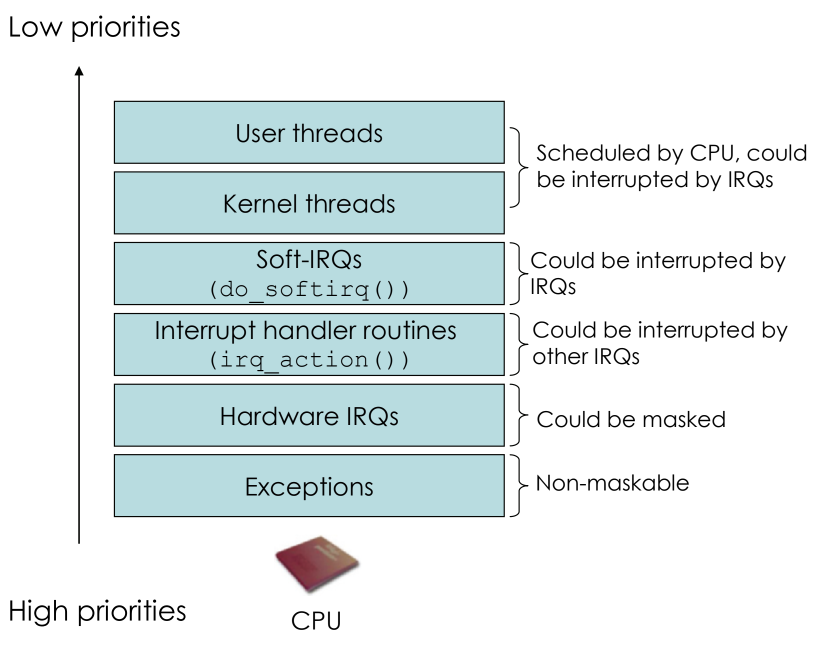

Low priorities
User threads
Kernel threads
Scheduled by CPU, could be interrupted by IRQs
Soft-IRQs (do_softirq())
Could be interrupted by IRQs
Interrupt handler routines (irq_action())
Could be interrupted by other IRQs
Hardware IRQs
Could be masked
Exceptions
Non-maskable
High priorities
CPU

Figure 18. Execution priorities in an embedded Linux kernel

handled by the ISR is masked during the top half of ISR. After the top half, the bottom half, i.e. `do_softirq()`, is invoked immediately to reduce the total ISR execution time. The interruption processed by the top half is enabled during the bottom half. Versions 2.4 and above of Linux are preemptive kernels. Hence, the scheduler is invoked to pick up the highest priority task to run, after the ISR. Kernel threads gain higher priority than user threads in Linux. If an external IRQ has the highest physical priority, and the control program for that event is implemented at the ISR level, then the interruption latency can be determined easily. Otherwise, the worse case interruption time from the interruption occurring to when the specific program is invoked must consider other interrupt sources and other executing processes on the CPU. Factors determining the interruption latency include: the use of preemptive or non-preemptive kernels; use of single or split ISRs; interruption enable/disable period, physical parameters such as the CPU context saving time, and the time of the longest CPU instruction.

Interruptions might occur very frequently. For instance, a network packet arrival to a network interface card can generates considerable workload for an embedded system in handling external events. This phenomenon leads to interruption overloading that might starve other important tasks running on the CPU. Several techniques can be applied to prevent interruption overload (Regehr and Duongsaa, 2005). The first technique is to disable the interruption for a period. System designers set either this period or its inverse, the interruption frequency, and then can determine the maximum interruption frequency that an embedded system would like to handle. The second method is to activate the interruption frequency control mechanism when the interruption overload is observed. Alternatively, hardware that schedules or arbitrates the I/O interrupts can be implemented to ease the interruption loading. As well as the embedded software improvement, hardware can also be specifically designed to reduce the interruption latency. ARM CPUs support fast interrupt (FIQ) architecture providing more banked registers than normal mode or other operation modes (Furber, 2000). FIQ is generally designed to support a direct memory access (DMA) transfer. The ARM CPU offers sufficient private registers to remove the need to save registers in applications that perform data transfer, thereby improving the response time and minimizing context switching overhead.

4.4 Embedded Device Driver and Power Management

Embedded systems, such as handsets, networked sensors and battery operated devices, are power sensitive. Previous studies indicate that the energy consumed by peripheral devices contributes a significant portion of the total power consumption of an embedded system. For instance, the LCD (Liquid Crystal Display) and wireless interfaces of a wireless communication PDA can consume 50% to 70% of the total energy in active mode (Nakamoto, 2004). For networked sensors, the radio and other peripheral devices consume more than 20mA, and a microprocessor consumes only 4mA during active mode (Hill et al., 2000). The peripheral devices consume more energy than the CPU, even in the inactive mode. Therefore, the power management of the embedded device driver should elaborate the low-power hardware designs and facilitate upper-layer applications to control the power usage. A number of research projects are working on a power-aware scheduler for embedded systems, concerning mainly the CPU resource and considering the current work load to adjust CPU speeds. One possible method is to utilize CPU dynamic voltage/clock scaling and change the CPU clock or voltage according to the current work load, thus saving power and achieving the performance requirement. Like the CPU design, the peripheral devices provide various operation modes, and each mode implies different numbers of active hardware components. For instance, the IEEE 802.11 WLAN network interface card supports at least two modes, the continuous access mode (CAM), which is always on and can achieve the best performance, and the power saving mode (PSM), which only wakes up if packets are transmitting or need to receive. PSM offers fewer throughputs than CAM. LCD also supports different operation modes, which provide the LCD different level backlights and brightness. Based on the hardware designs, the device driver can thus employ the hardware low-power features to export flexible power management functions to upper-layer EOS or embedded software (Vaddagir et al., 2004). Moreover, device drivers or system modules can implement low-power schemes to reduce energy consumption. The schemes in device drivers or system modules further optimize the power consumption for particular applications or usage models. For instance, the device can dynamically switch on and off, or set itself to different modes, according to the usage pattern. Also, a device driver or I/O subsystem might need to reschedule or merge I/O requests to eliminate redundant requests to save energy. For instance, the previous study optimized power consumption for TCP/IP over WLAN, web access over WLAN and voice over WLAN. These designs can be implemented on device drivers or system modules.

The power management interface of an EOS is also important. The APIs (Application Programming Interfaces) generalize the power management features that are provided by peripheral devices, and offer a single interface to an EOS. Advanced Configuration and Power Interface (ACPI) specification is well-known power management interface defined for general-purpose PCs, but only provides static power management which suspends/resumes the devices. IBM and MontaVista Software have jointly developed a dynamic power management (DPM) architecture for embedded systems, enabling the embedded software to optimize the power according to its needs (IBM and MontaVista, 2002). Figure 19 shows a possible implementation of dynamic power management on an embedded system. The CPU and peripheral devices physically support low-power functions at the lowest layer. The power-aware device drivers and a power-aware CPU scheduler are implemented based on the hardware features. A particular CPU mode and a peripheral device mode are combined to form a policy. This policy design benefits upper-layer EOS and embedded applications to manage the power-aware embedded system easily. The policy manager implemented in the EOS kernel wraps the policies to upper-layer applications. Therefore, embedded applications can employ DPM APIs to optimize the power consumption according to their own needs. The DPM strategies for an embedded application are the power management method particularly for the embedded application. An embedded application typically has pre-determined behaviors, so the power consumption can usually be predicted and fully controlled in all situations. The power management strategy can be realized by investigating all application running states, each having its own CPU and peripheral modes, and calling the policy management to change the modes dynamically. The dotted line shown in

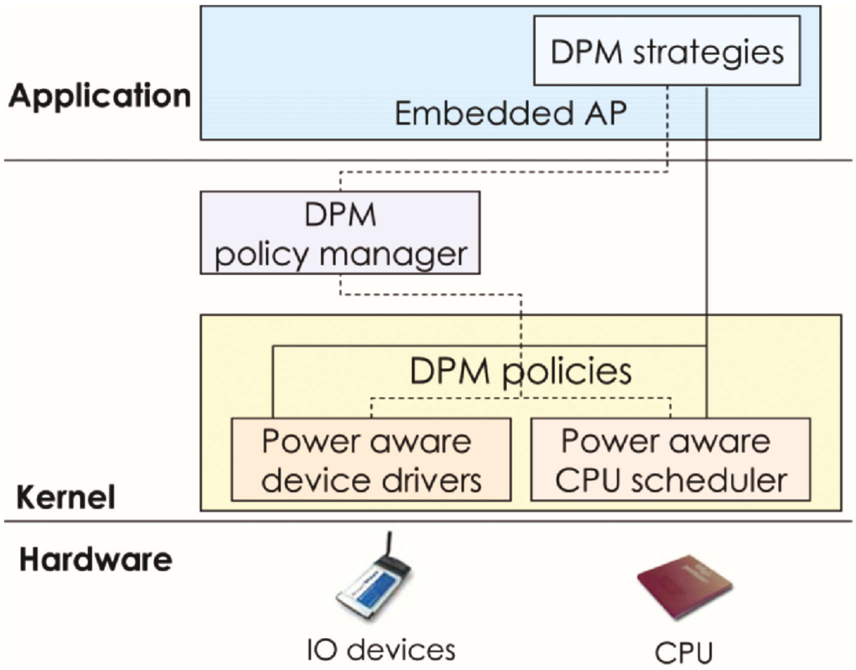


Figure 19. Implementation of dynamic power management of embedded software

Figure 19 denotes the flow of the DPM operations. EOSs that do not implement DPM and have their own power management APIs can be directly invoked by the embedded application. The power management APIs can be employed to minimize the power consumption of these embedded systems.

5. EMBEDDED SOFTWARE TOOLCHAIN

With the benefits of shorter time-to-market and future modifiability and extensibility, a lot of designs in the embedded systems have shifted to the software side. Software development tools, such as compilers, linkers, loaders, assemblers, debuggers, and simulators, have become an integral part of the embedded system design.

This section begins with an overview of the embedded software development toolchains, including compilers, linkers, loaders, debuggers, and simulators. It is then followed by the discussion of some important issues when compiling code for embedded systems.

5.1 Overview of the Embedded Software Development Toolchains

Development tools such as compilers, assemblers, debuggers, and simulators are no strangers to software developers. Unlike tools for general-purpose computing systems, the embedded software tools are often for *cross development*. In a cross development environment, the system where software is developed (usually called the *host system*) is different from the target system on which the developed software will run. The target embedded processor is usually not appropriate for software development due to lack of user-friendly OS, software interfaces, and/or limited hardware resources.

The basic structure of a typical software development toolchain is shown in Figure 20. Programs written in high-level languages (usually C) or assembly languages are compiled or assembled into object files, which, along with some libraries, are linked together to produce executable files. The generated executable files can be executed through a simulator on the host machine. Or they can be loaded to the actual target device and run there. During the course of the development process, the executables can also run under a debugger (which can run on the actual target device or through a simulator as well).

Compilers

Compilers have long been considered the most important tool in software development on general-purpose computing systems. In the embedded system domain, compilers were not as important in the past because it was often necessary to write applications in assembly languages due to embedded processors' special instruction sets, tight code size constraints, and performance concerns. However, with the increasing complexity of applications and the growing popularity of general programmable processors, more and more embedded applications and algorithms are implemented in high-level languages to avoid time-consuming and error-prone assembly programming. Compilers, especially optimizing compilers, are therefore becoming a key component in embedded software development.

Figure 21 depicts the basic structure of a typical optimizing compiler. A compiler normally consists of two major pieces: a front-end and a back-end. The front-end translates high-level languages to compiler internal representations (IR), on which the subsequent components of the compiler will operate. A compiler might have multiple front-end modules, one for each high-level language that it supports. The back-end performs optimization and generates target machine code (or assembly code). The optimizations performed by the back-end can be further classified into two categories: (1) high-level optimizations (HLO), which deal with machine-independent optimizations such as loop transformations, dead code elimination,

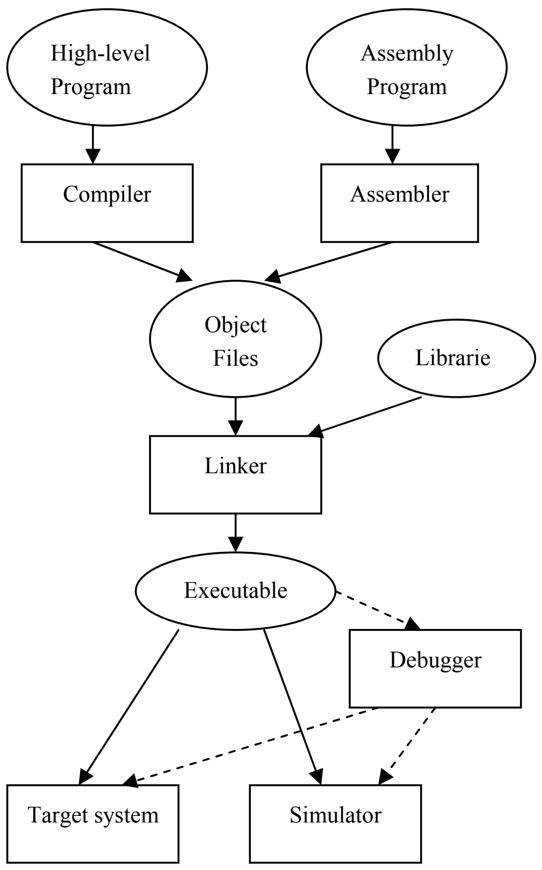


Figure 20. Basic structure of a typical software development toolchain

copy/constant propagation, among others, and (2) low-level optimizations (LLO), which focus on machine-dependent optimizations such as instruction scheduling, register allocation, and any other code transformation that relies on the knowledge of the target machine architecture. Details for the compiler structure and common code generation and optimization techniques can be found in various compiler books (Aho et al., 1986; Cooper and Torczon, 2004; Muchnick, 1997).

Linkers and loaders

Linkers and loaders perform highly-related but different works (Levine, 1999). Linkers combine object files (generated by compilers or assemblers)

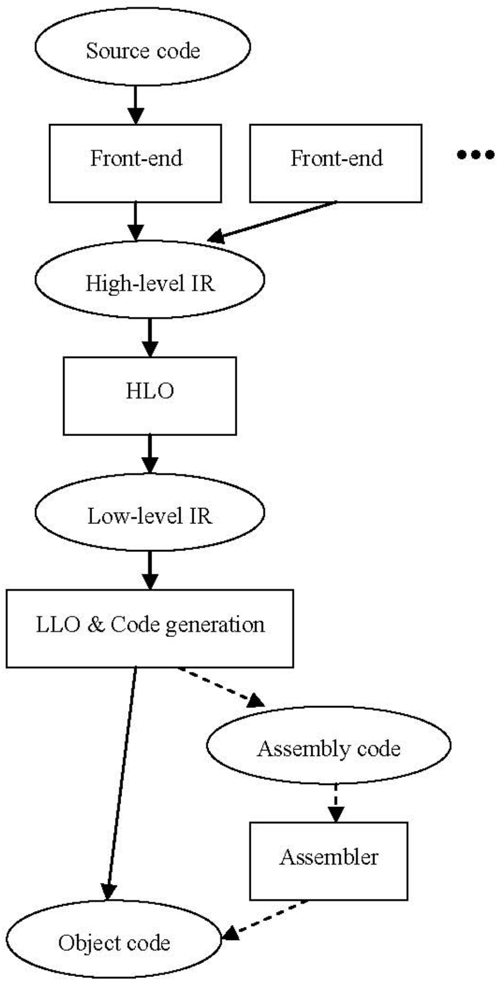


Figure 21. Basic structure of a typical optimizing compiler

and library files into executables. When combining object files, a linker resolves symbol references, verifies that all external references are satisfied, and performs relocation, among other duties. In some of the memory-constrained embedded systems where overlays are required, the linker's job is more complicated than the case where virtual memory is supported.

Linkage can happen *statically* or *dynamically*. For static linking, all constituent components of an executable must be present at link time and there cannot be unresolved external references. (Such an executable is sometimes called an *archive-bound* executable.) On the other hand, dynamic linking defers resolution of some external references until run time.

Although dynamic linking is faster at link time and can support better code sharing at run time, it is rarely used in the embedded software due to the substantial performance cost and potential run time errors that may not show up in testing.

In a software development toolchain, the linker is often the only component that can see all pieces of a program (as most of the compilers only work on a single source file at a time). Therefore the linker becomes a idea place to perform whole-program analysis and optimization. While performing optimization at link time has the advantages of seeing the whole program and being able to handle code without source files, due to lack of source code information, the link-time optimizer often cannot be as aggressive as the compiler in certain optimizations that require the knowledge of the source program structures. The optimizations that show great promise at link time include dead code/data removal, branch optimization, calling convention optimization, data layout optimization, and so on (Srivastava and Wall, 1994; Haber et al., 2003; De Bus et al., 2004).

Loaders, usually part of the OS, bring a program from secondary storage into main memory, sometimes with relocation, so that the program can run. Unlike in the general-purpose computing systems where handling dynamically linked libraries is considered one of the most important jobs for loaders, in the embedded systems, loaders usually focus more on the areas such as decompression of the code and data.

Debuggers

During the course of a software development project, generally more time is spent in testing and debugging than in code writing. With good use of debuggers, which allow the developers to examine and modify the state of a running program, the software development time can be shortened substantially.

In the embedded development environment, the debugger usually runs on the host machine while the debugged software runs on the target system. This is called *cross debugging* (as opposed to native debugging where the debugger and the software are running on the same machine). Cross debugging is considered more difficult than native debugging. In cross debugging, besides the necessity of pre-negotiated communication protocols between the target system and the debugger on the host system, the target processor (and sometimes the system board) often needs to provide the architectural supports for debugging as well.

Here is an example that shows the difficulty with cross debugging. When setting a breakpoint in a native debugger, the normal practice is for the debugger to replace the instruction where the breakpoint is set with a hardware break instruction (or a branch instruction that jumps to a breakpoint handling routine). However, in an embedded system, if the code is in read-only memory (ROM), instructions cannot be replaced at run time. One way to solve this problem is to use a system debugging tool called *In-Circuit Emulator* (ICE). Basically the emulator substitutes the processor in the target system (sometimes with the target processor being removed from the system board), replicates the processor's operations, and provides the ability to examine and change the contents of registers, memory and I/O. Normal arrangement for an ICE is shown in Figure 22, where the ICE is plugged into the target system (board) on one side and connected to the host system on the other. When working with an emulator, the debugger does not need to change the code for a user-set breakpoint. Instead, the emulator stops the code when it sees the address (at which the breakpoint is set) is about to be executed.

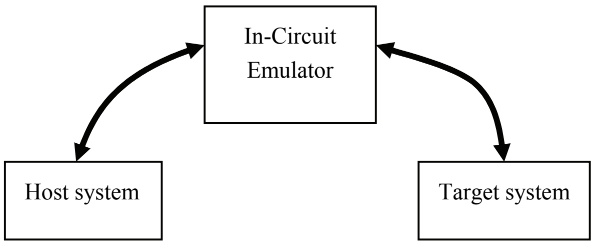


Figure 22. Normal in-circuit emulator connection

However, with the advent of SoC where the processor is not by itself an independent chip, it is very difficult (if not impossible) to replace the processor with an ICE. Several embedded processors, such as ARM (Furber, 2000), have therefore added architectural features that provide debug supports comparable with what was offered by an ICE.

Simulators

Simulators are important tools for both software developers and architecture designers alike. In the course of embedded system design, a simulator can support software testing and verification even before the target silicon exists. It can also provide an experiment platform for the architecture designers to explore design alternatives.

The accuracy and granularity of information provided by simulators can range from detailed timing analysis, cycle accurate simulation, to behavioral simulation. Normally (and intuitively), the more detailed information a simulator provides, the slower it runs.

When writing simulators, in particular instruction-set simulators, one can choose to use interpreted or compiled simulation (Fisher et al., 2005). Traditional interpreted simulation (such as widely-used SimpleScalar simulator (2004)) is flexible, straightforward, and easy to implement, but, due to its interpretive nature, is very slow. In an interpreted simulation, the simulator is basically an interpreter with a main loop that fetches, decodes and executes (simulates) instructions from the simulated application program one by one, as shown in Figure 23 (Reshadi et al., 2003).

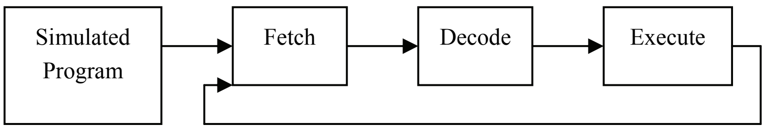


Figure 23. Interpreted simulation work flow

Compiled simulator, on the other hand, runs a lot faster but lacks flexibility. There have been several compiled simulation techniques proposed in the past (Reshadi et al., 2003; Živojnović et al., 1995; Pees et al., 1997; Maurer and Wang, 1991; Nohl et al., 2002). Conceptually, in a compiled simulation (as shown in Figure 24), the simulated application program is first decoded and translated into another program in high-level language (most likely C). The translated program is then compiled into host native code by the compiler on the host machine. Executing this host native program *is* a simulation of the application program running on the target system.

5.2 Important Issues for Embedded Compilers

While the embedded compilers are in general no different from the general-purpose compilers structurally, there are nonetheless several compilation issues specific to the embedded compilers. These issues are understandably all in the backend (especially machine dependent) modules. For general-purpose compilers, run-time performance is probably the foremost concern when generating and optimizing code. However, in the embedded world, speed is not the only concern (sometimes not even the most important goal) for compilers. Due to the hardware resource constraints and time-to-market pressure, other factors such as power consumption, code size, and retargetability are as important (if not more) as

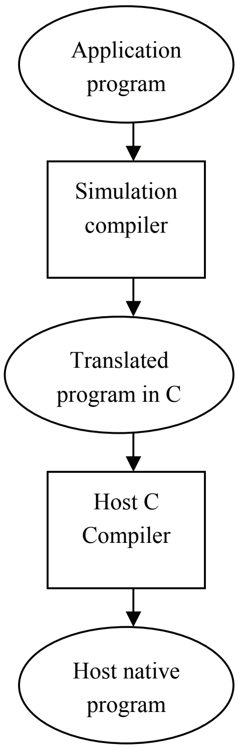


Figure 24. Compiled simulation work flow

performance when generating embedded code. While many classic optimization techniques can equally benefit performance, power, and code size (e.g. common subexpression elimination (Muchnick, 1997) would reduce power and code size while speeding up the code), a lot of time these goals are conflicting and the compiler designers (or even the compiler users) will need to make tradeoff. In fact, in order to strike a better balance among these conflicting considerations, integer linear programming is widely used in code generation and optimization (especially in instruction selection/scheduling and register allocation) for embedded compilers (Kessler and Bednarski, 2002; Naik and Palsberg, 2002; Kong and Wilken, 1998).

In the rest of this subsection we will look at some compilation issues specific to the embedded systems.

Code size reduction

In the embedded systems, software often has to run under constrained memory resource. Smaller code footprints can also have a positive effect on the (instruction) cache performance. Reducing the code size of application programs has therefore become one of the important compilation goals. In general, code size can be reduced either through optimization or code compression.

1. *Optimization*

A lot of classic scalar optimization techniques focus on reducing code size. Most of these techniques are trying to remove dead, unnecessary, or redundant code from the program. Among some of the most well-known and straightforward optimizations are dead/unreachable code removal, common subexpression elimination, copy/constant propagation, and strength reduction (Muchnick, 1997).

Besides removal of dead and redundant code, other advanced optimizations were proposed to better utilize the architecture features in order to reduce the number of instructions generated. For example, instruction selection in code generation phase often plays an important role in final code size as there could be several different code sequences to choose from when generating code for a particular operation. Another architecture feature that can be exploited to cut down code size is the auto-increment (or auto-decrement) addressing mode of memory operations. In unoptimized code, each memory access instruction (i.e. load/store) normally requires an arithmetic instruction to set up its address, as shown in the example in Figure 25 (a). In this example, both load instructions (I2 and I4) need arithmetic instructions (I1 and I3) to set up their addresses. If the target architecture supports auto-increment/decrement addressing mode, the compiler can arrange for the first load (I2) to modify its address register through auto-increment so that the modified address register value corresponds to the memory address used by the second instruction (I4), as shown in Figure 25 (b). This way the arithmetic instruction that calculate the second load's address is no longer needed and can be removed.

```
I1: add a = t + 4          I1: add a = t + 4
I2: load x = *(a)          I2: load x = *(a)+
I3: add b = t + 8          I4: load y = *(a)
I4: load y = *(b)

         (a)                        (b)
```

Figure 25. Exploit auto-increment addressing mode to reduce code size

Given a code sequence like the example in Figure 25 (a), it is relatively easy to convert memory instructions to the ones with auto-increment/decrement addressing mode. A more difficult problem is how to allocate memory locations to variables so that we can maximize the opportunities of performing auto-increment/decrement optimization. This storage allocation problem is called *offset assignment* problem (Liao et al., 1996). One popular solution is to model the variable access order as a graph and the objective is to find the maximum weight path cover (Liao et al., 1996). This approach has been enhanced later to reduce cost or improve performance (Rao and Pande, 1999; Zhuang et al., 2003).

Note that in the cases where code size is the most important factor for compiling the applications, optimization techniques, such as loop unrolling, procedure inlining, tail duplication, etc., will need to be turned off as they only benefit performance but hurt code size tremendously.

2. *Code Compression*

Reduction in code size can also be achieved by compressing the code. While there are many popular file-level data compression techniques, they are not suitable for code compression on embedded processors as the programs can have branches/jumps which often require us to be able to decompress the code at any point. Many code compression schemes proposed are dictionary based (Lekatsas and Wolf, 1998; Lefurgy et al., 1997; Thuresson and Stenstrom, 2005). In a dictionary based compression scheme, a program is analyzed statically and the instructions that appear more frequently are identified and replaced with *codewords* that are much smaller than the original instructions. At run-time, a codeword fetched is first used for referencing the dictionary and then replaced by the original instruction recorded in the dictionary.

Decompression can happen before the code is brought into the cache (Wolfe and A. Chanin, 1992; Lekatsas and Wolf, 1998). This way the instruction fetch unit of the processor still fetches the normal uncompressed code and the decompression only happens when there is an instruction cache miss. Or decompression can happen in the decode stage in the processor pipeline (Ros and Sutton, 2003) as depicted in Figure 26. In this scheme, when an instruction is fetched, the hardware needs to determine whether to send the instruction to the decompressor if it is compressed, or whether to simply pass it on to the decode stage.

Several embedded architectures, such as ARM (Furber, 2000) and MIPS (MIPS Technologies, 2001), provide instruction set support for code compression. For example, the ARM architecture, which is arguably the most popular embedded architecture, supports 32-bit RISC instruction set.

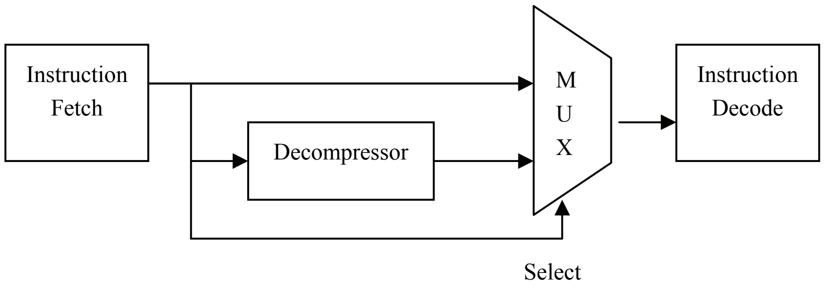


Figure 26. Datapath for code decompression

Most of the recent implementations of the ARM architecture support a 16-bit instruction set extension called *Thumb*. The Thumb instruction set provides the most commonly used ARM instructions in 16-bit format. The instructions are dynamically decompressed (into normal ARM instructions) in the processor pipeline. How the Thumb decompressor fits in the pipeline is very similar to the datapath diagram shown in Figure 26.

Unfortunately the use of Thumb instructions doesn't come without cost. For a program, the dynamic count of instructions executed increases when Thumb instructions are used. This usually leads to longer execution time. Some study was done on how to better generate mixed ARM and Thumb code guided by profile information so that the code size reduction can be achieved without loss in performance (Krishnaswamy and Gupta, 2002).

Optimization for low power

Reducing energy usage has always been an important challenge for the embedded system designers. The approaches to address energy problems comprise techniques ranging from circuit design, architecture design, operating system support, to compiler support. While the power saving schemes based on hardware design and/or OS supports have been working effectively, it is equally important to design energy-aware compilation techniques in order to achieve the best energy-efficient systems.

In CMOS circuits, power dissipation is proportional to the square of input voltage (Chandrakasan and Brodersen, 1995). Therefore any reduction in the input voltage will have quadratic effect on energy saving. However, reducing voltage would have a negative impact on the clock frequency and cause a program to run slower. A lot of software based approaches have therefore focused on when (or where) to scale (reduce) the input voltage for the running programs in a system so that the timing constraints can still be met. While the operating system appears to be a better place to control the voltage scaling (due to its ability to see all the running processes in the system), compiler based approaches can often provide finer granularity of control. For example, Hsu and Kremer (2002) proposed techniques that can identify program regions where the processor can be slowed down. The speed for each region is set accordingly. Saputra et al. (2002) proposed two energy optimizations based on both static and dynamic voltage scaling. In static scaling, loop-level optimization technique was leveraged to create opportunities for voltage scaling. In dynamic scaling, integer linear programming was exploited to select different supply voltage for different parts of the code to accommodate both energy and performance constraints. AbouGhazaleh et al. (2003) presented a hybrid compiler/OS scheme for dynamic voltage scaling. In their approach, power management hints are inserted in the application code by the compiler so that the operating system has fine-grained, path-specific information to adjust processor performance and energy consumption.

Several phases of the compiler backend could be easily made more energy conscious (instead of only focusing on the performance). For example, the cost functions and heuristics that are used for instruction selection and scheduling during code generation could be modified to take into account the power consumed by each instruction (or instruction sequence).

Retargetable compilers

Unlike their counterparts in the desktop domain, embedded processors have far more variations and flavors in the architecture design. Even for the same instruction set architecture (ISA) family, new features and instructions (such as multi-media or network extensions) are constantly added to the existing architecture to meet the demands spawned from the ever-changing new applications in the embedded world. In addition, the microarchitecture is frequently changed to take advantage of the rapid advancement in semiconductor process technology. Such changes, however small, would often leave the existing software development tools, especially compilers, inadequate or even unusable. The embedded processor vendors are therefore forced to re-design and re-implement the compilers. Such practice is unfortunately time-consuming and often too costly. It is hence desirable (if not imperative) to have an easily retargetable compiler that can quickly adapt to a new architecture extension or a microarchitecture enhancement.

In the heart of a retargetable compiler is the architecture/machine description language (ADL). The backend of a retargetable compiler is usually made orthogonal (independent) to any specific architecture and driven by the ADL. In fact, other components of the development toolchain, such as assemblers and simulators, can also be made ADL driven. With this approach, whenever a change is made to the ISA or microarchitecture, the tool designers need only to modify the machine descriptions written in the provided ADL and the new development tools will be quickly available. The tool design cost, as a result, will be greatly reduced.

Several ADLs have been proposed to support retargetable toolchains. Among the most famous and widely used are LISA from Aachen University of Technology in Germany (Pees et al., 2000), EXPRESSION from University of California, Irvine (Halambi et al., 1999), and nML developed at Tech. University Berlin (Fauth et al., 1995). Some popular general-purpose compilers, such as GNU gcc compiler and IMPACT compiler from University of Illinois (Gyllenhaal et al., 1996), have their own machine description languages as well to drive their compiler backend. One thing to keep in mind is that it is almost impossible to design an ADL that can properly describe and model architectures across all application domains. An ADL is easier to design if it is targeted only at processors specialized for a particular application domain.

A lot of retargetable compilers are part of bigger hardware/software co-design projects (Halambi et al., 1999; Lanneer et al., 1995; Hoffmann et al., 2001; Aditya et al., 1999). The methodology of co-design approaches often requires iterative process of hardware modification, co-simulation, and system evaluation until design criteria are met. Retargetable compilers are therefore fundamental in this co-design scheme in order for different design choices to be quickly explored and evaluated.

6. CONCLUSIONS AND FUTURE DIRECTIONS

With the requirement of shorter time-to-market and feature modifiability and extensibility, many designs in the embedded systems have shifted to the software side. Conventional software development mythology cannot be applied to embedded software directly because embedded software works tightly with specific hardware. In this chapter, we discussed the most important components of embedded software, including low-power task scheduling, low-power device scheduling, the development framework for device drivers, and embedded software toolchain.

DVS is widely used to speed down the processor and reduce its energy consumption. A real-time DVS algorithm minimizes energy consumption while keeping timing constraints. Section 2 first described a list of mechanism to use slack time and next introduced several referred real-time DVS scheduling algorithms. Each algorithm is unique in its way of calculating available slack time and allocating slack to available tasks. In addition, the tradeoff between energy-saving performance and complexity was also well studied. To further enhance the performance of real-time DVS, many issues remain to be addressed, such as the impact of data dependency, static current leakage, and context switches. Finally, an efficient integration of DVS and DPM deserves more research efforts to consider all aspects of power consumption from a system-wide viewpoint.

DPM is a powerful methodology for reducing energy consumption in modern embedded systems where peripheral devices consume more energy than the processor. Section 3 introduced and classified existing DMP policies and latest development. Due to the extreme complexity of this problem, there are still many issues left to be solved in designing an efficient real-time DPM policy. Finally, a complete energy-profiling tool at the level of an operating system is needed for designers to collect power-performance statistics on real systems. The availability of this data is essential in the design of the next-generation power management mechanism.

Section 4 introduced the impact of I/O devices and their drivers on the functionalities, performance, and efficiency of an embedded system. The hardware/software co-design method saves the development time of the embedded device, but there are still many issues, such as methodologies and tools for HW/SW co-design and HW/SW co-simulation, needed to be resolved. Regarding device driver development, two approaches, i.e. non-OS-based and EOS-based, are commonly used. The non-OS-based implementation is more efficient and requires less memory than the EOS-based approach, but embedded software based on the EOS-based implementation utilizes the services offered by an EOS, and thus can reduce the development time and the complexity of the embedded software significantly. System designers should tradeoff these parameters during the driver design and implementation. Besides the above development issues, further research on device driver frameworks and interrupt handling processes in EOSs is highly required to guarantee the real-time characteristics especially for hard real-time systems.

Software development tools are instrumental in the successful roll-out of an embedded system. While several tools such as assemblers, debuggers, and linkers are generally considered mature technology, they actually deserve more attention in the research community so that the design process of embedded software can be made faster and more smoothly. Compilers, on the other hand, have garnered a lot of interests recently in the academic and industrial communities alike due to the shift from assembly languages to high-level languages in embedded software development. Besides the important research topics such as energy saving, code size reduction, and retargetability mentioned in Section 5, other compilation issues, such as how to strike a good balance among various code generation factors and how to work better with the architecture and OS supports, still need to be further studied and investigated.

REFERENCES

AbouGhazaleh, N., Childers, B., Mosse, D., Melhem, R., and Craven, M., 2003, Energy management for real-time embedded applications with compiler support, in *Proceedings of the 2003 ACM SIGPLAN Conference on Language, Compiler, and Tool for Embedded Systems*, pp. 284-293.

Acquaviva, A., Benini, L., and Ricco, B., 2001, Software-controlled processor speed setting for low-power streaming multimedia, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, **20**(11), pp. 1283-1292.

Aditya, S., Rau, B. R., and Kathail, V., 1999, Automatic architectural synthesis of VLIW and EPIC processors, in *Proceedings of the 12th international symposium on System synthesis*, p.107.

Aho, A. V., Sethi R., and Ullman J. D., 1986, *Compilers: Principles, Techniques, and Tools*, Addison Wesley.

Anand, M., Nightingale, E. B., and Flinn, J., 2004, Ghosts in the machine: interfaces for better power management, in *Proceedings of the second international Conference on Mobile systems, Applications, and Services.* pp. 23-35.

Aydin, H., Melhem, R., Moss´e, D. and -Alvarez, P. M., 2001, Dynamic and aggressive scheduling techniques for power-aware real-time systems, in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pp. 95-105.

Aydin, H., Melhem, R., Mosse, D., and -Alvarez, P. M., 2004, Power-aware scheduling for periodic real-Time tasks, *IEEE Trans. on Computers,* **53**(5), pp. 584-600.

Barned, R. M., and Richards, R. J., 2002, Uniform Driver Interface (UDI) reference implementation and determinism, *Proceedings of Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 301-310.

Benini, L., Bogliolo, A., Paleologo, G. A., and Micheli, G. D., 1999, Policy optimization for dynamic power management, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, **18**(6), pp. 813-833.

Benini, L., Bogliolo, A., and Micheli, G. D., 2000, A survey of design techniques for system-level dynamic power management, *IEEE Trans. VLSI Systems*, **8**(3), pp. 299-316.

Brock, B., and Rajamani, K., 2003, Dynamic power management for embedded systems. In *Proceedings of the 2003 IEEE International SOC Conference.*

Cai, L., and Lu, Y. H., 2005, Joint power management of memory and disk, in *Proceedings of Design, Automation and Test in Europe*, pp. 86-91.

Chandrakasan, A. P., and Brodersen, R. W., 1995, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, Norwell, MA.

Choi, I., Shim, H., and Chang, N., 2002, Low-power color TFT LCD display for hand-held embedded systems, in *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, pp. 112-117.

Choi, K., Soma, R., and Pedram, M., 2004, Dynamic voltage and frequency scaling based on workload decomposition, in *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, pp. 174-179.

Chu, E. T.-H., Huang, T. Y., Liu, K. Y., Tsai, C. H., and Chen, P. Y., 2006, COLORS: A real-time DPM policy with DVS support, submitted to *Proceedings of Design, Automation and Test in Europe*.

Chung, E. Y., Benini, L., and Micheli, G. D., 1999, Dynamic power management using adaptive learning tree, in *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pp. 274-279.

Cooper, K. D. and Torczon, L., 2004, *Engineering a Compiler*, Morgan Kaufmann.

De Bus, B., De Sutter, B., Van Put, L., Chanet, D., De Bosschere, K., 2004, Link-time optimization of ARM binaries, *ACM SIGPLAN Notices*, v.39 n.7, July.

Ethier, S., 2003, Application-driven power management, QNX Software Systems Ltd.

Fauth, A., Van Praet, J., and Freericks, M., 1995, Describing instruction set processors using nML, in *Proceedings of the 1995 European conference on Design and Test*, page 503.

Fisher, J. A., Faraboschi, P., Young, C., 2005, *Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools*, Morgan Kaufmann.

Furber, S., 2000, *ARM System-on-Chip Architecture*, 2nd ed., Addison Wesley.

Gyllenhaal, J. C., Hwu, W.-m. W., and Rau, B. R., 1996, HMDES version 2 specification, *IMPACT Technical report*, IMPACT-96-03, University of Illinois, Urbana IL.

Haber, G., Klausner, M., Eisenberg, V., Mendelson, B., and Gurevich, M., 2003, Optimization opportunities created by global data reordering, in *Proceedings of the International Symposium on Code Generation and Optimization*, pp. 228-237.

Halambi, A., Grun, P., Ganesh, V., Khare, A., Dutt, N., and Nicolau, A., 1999, EXPRESSION: a language for architecture exploration through compiler/simulator retargetability, in *Proceedings of the conference on Design, automation and test in Europe*, Article No. 100.

Hill, Jason, and et al., 2000, System architecture directions for networked sensors, *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*.

Hoffmann, A., Kogel, T., Nohl, A., Braun, G., Schliebusch, O., Wahlen, O., Wieferink, A., and Meyr, H., 2001, A novel methodology for the design of application specific instruction set processors (ASIP) using a machine description language, *IEEE Transactions on Computer-Aided Design*, 20(11):1338-1354.

Honda, S., and Takada, H., 2003, Evaluation of applying SpecC to the integrated design method of device driver and device, *Design, Automation and Test in Europe Conference and Exhibition*.

Hsu, C.-H. and Kremer, U., 2002, Single vs. multiple regions: A comparison of different compiler-directed dynamic voltage scheduling approaches, in *Proceedings of Power-Aware Computer Systems Workshop*.

HP, Intel, Microsoft, Phoenix, and Toshiba; http://www.acpi.info/

HP Labs and P. Alto, Power evaluation of a handheld computer, 2003, *Micro IEEE*, **23**(1), pp. 66-74.

Hwang, C. H., and Wu, A. C., 1997, A predictive system shutdown method for energy saving of event-driven computation, in *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pp. 28-32.

IBM and MontaVista Software, 2002, Dynamic Power Management for Embedded System; http://www.research.ibm.com/arl/publications/papers/DPM_V1.1.pdf.

Irani, S., Shukla, S., and Gupta, R., 2003, Online strategies for dynamic power management in systems with multiple power-saving states, *ACM Trans. on Embedded Computing Systems*, **2**(3), pp. 325-346.

Ishihara, T.,and Yasuura, H., 1998, Voltage scheduling problem for dynamically variable voltage processors, in *Proceedings of ACM International Symposium on Low-Power Electronics and Design*, pp. 197-199.

Jejurikar, R., and Gupta, R. K., 2004, Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems, in *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, pp. 78-81.

Jejurikar, R., Pereira, C., and Gupta, R. K., 2004, Leakage aware dynamic voltage scaling for real-time embedded systems, in *Annual ACM IEEE Design Automation Conference*, pp. 275-280.

Jejurikar, R., and Gupta, R. K., 2004, Procrastination scheduling in fixed priority real-time systems, *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools in Embedded Systems*, pp. 57-66.

Jerraya, A., and et al., 2003, *Embedded Software for SOC*, Kluwer Academic Publishers, Norwell, MA.

Karlin, A., Manasse, M., McGeoch, L., and Owicki, S., 1994, Competitive randomized algorithms for nonuniform problems, *Algorithmica*, **11**(6), pp. 542-571.

Kessler, C. W., and Bednarski, A., 2002, Optimal integrated code generation for clustered VLIW architectures, in *Proceedings of the 2002 Joint Conference on Languages, Compilers, and Tools for Embedded Systems & Software and Compilers for Embedded Systems*, pp. 102-111.

Kim, M., and Ha, S., 2001, Hybrid run-time power management technique for real-time embedded system with voltage scalable processor, in *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, pp. 11-19.

Kim, W., Kim, J., and Min, S. L., 2002, A dynamic voltage scaling algorithm for dynamic priority hard real-time systems using slack time analysis, in *proceedings of the conference on design, automation and test in europe*, pp. 788-794.

Kim, W., Kim, J., and Min, S. L., 2003, Dynamic voltage scaling algorithm for fixed priority real-time systems using work-demand analysis, in *proceedings of the 2003 international symposium on low power electronics and design*, pp. 396-401.

Kong, T. and Wilken, K. D., 1998, Precise register allocation for irregular architectures, in *Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture*, pp. 297-307.

Krishna, C. M., and Lee, Y. H., 2003, Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems, *IEEE Trans. on Computers*, **52**(12), pp. 1586-1593.

Krishnaswamy, A. and Gupta, R., 2002, Profile guided selection of ARM and thumb instructions, in *Proceedings of the 2002 Joint Conference on Languages, Compilers, and Tools for Embedded Systems & Software and Compilers for Embedded Systems*, pp. 56-64.

Krishnapura, R., Goddard, and Qadi, A., 2004, A dynamic real-time scheduling algorithm for reduced energy consumption, *Technical Report TR-UNL-CSE-2004-0009*, University of Nebraska Lincoln.

Labrosse, Jean J., 2002, *MicroC OS II: The Real Time Kernel*, CMP Books.

Lanneer, D., Van Praet, J., Kifli, A., Schoofs, K., Geurts, W., Thoen, F., and Goossens, G., 1995, CHESS: Retargetable code generation for embedded DSP processors, in *Code Generation for Embedded Processors*, Kluwer Academic Publishers Norwell, MA, pp. 85-102.

Lee, C. H., and Shin, K, G., 2004, On-line dynamic voltage scaling for hard real-time systems using the EDF algorithm. In *Proceedings of the 25th IEEE Real-Time Systems Symposium*.

Lefurgy, C., Bird, P., Chen, I-C., and Mudge, T., 1997, Improving code density using compression techniques, in *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pp. 194-203.

Lekatsas, H. and Wolf, W., 1998, Code compression for embedded systems, in *Proceedings of the 35th annual conference on Design automation*, pp. 516-521.

Levine, J. R., 1999, *Linkers and Loaders*, Morgan Kaufmann.

Liao, S., Devadas, S., Keutzer, K., Tjiang, S., and Wang, A., 1996, Storage assignment to decrease code size, *ACM Transactions on Programming Languages and Systems*, Volume 18, Issue 3, pp. 235-253.

Li, Qing, and Yao, Caroline, 2003, *Real-Time Concepts for Embedded Systems*, CMP Books.

Liu, C. L., and Layland, J., 1973, Scheduling algorithms for multiprogramming in a hard real-time environment, *Journal of the ACM*, 10(1), pp. 46-61.

Liu, J., and Chou, P. H., 2004, Optimizing mode transition sequences in idle intervals for component-level and system-level energy minimization, in *Proceedings of 2004 International Conference on Computer Aided Design*, pp. 21-28.

Lu, Y. H., Benini, L., and Micheli, G. D., 2000, Low-power task scheduling for multiple devices, in *Proceedings of the eighth international workshop on Hardware/software codesign*, pp. 39-43.

Lu, Y. H., and Micheli, G. D., 2001, Comparing system-level power management policies, *IEEE Design & Test of Computers*, **18**(2), pp. 10-19.

Massa, Anthony J, 2002, *Embedded Software Development with eCos*, Prentice Hall.

Maurer, P. M., and Wang, Z., 1991, Techniques for unit-delay compiled simulation, in *Proceedings of the 27th Conference on Design Automation*, pp. 480-484.

Microsoft; http://www.microsoft.com/whdc/system/pnppwr/powermgmt/devicepm.mspx

Mochocki, B., Hu, X. S., and Quan, G., 2005, Practical on-line DVS scheduling for fixed-priority real-time system, in *the 11th IEEE Real-Time and Embedded Technology and Applications Symposium.*

MIPS Technologies, 2001, *MIPS32 Architecture for Programmers Volume IV-a: The MIPS16 Application Specific Extension to the MIPS32 Architecture*, March.

Muchnick, S. S., 1997, *Advanced Compiler Design and Implementation*, Morgan Kaufmann.

Naik, M. and Palsberg, J., 2002, Compiling with code-size constraints, in *Proceedings of the 2002 Joint Conference on Languages, Compilers, and Tools for Embedded Systems & Software and Compilers for Embedded Systems*, pp. 120-129.

Nakamoto, Y., 2004, Toward mobile phone Linux, *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 117-124.

Nohl, A., Braun, G., Schliebusch, O., Leupers, R., Meyr, H., and Hoffmann, A., 2002, A universal techniques for fast and flexible instruction-set architecture simulation, in *Proceedings of the 39th Conference on Design Automation*, pp. 22-27.

Pees, S., Živojnović, V., Ropers, A., and Meyr, H., 1997, Fast simulation of the TI TMS 320C54x DSP, in *Proceedings of the International Conference on Signal Processing Applications and Technology*, pp. 995-999.

Pees, S., Hoffmann, A., and Meyr, H., 2000, Retargeting of compiled simulators for digital signal processors using a machine description language, in *Proceedings of the conference on Design, automation and test in Europe*, pp. 669-673.

Pillai, P., and Shin, K. G., 2001, Real-time dynamic voltage scaling for low-power embedded operating systems, in *proceedings of the eighteenth ACM symposium on Operating systems principles*, pp. 89-102.

Quan, G., and Hu, X. S., 2001, Energy efficient fixed-priority scheduling for real time systems on variable voltage processors, in *Annual ACM IEEE Design Automation Conference*, pp. 828-833.

Quan, G., and Hu, X. S., 2002, Minimum energy fixed-priority scheduling for variable voltage processors, in *Proceedings of the conference on Design, automation and test in Europe,* pp. 782.

Quan, G., Niu, L., Hu, X. S., and B. Mochocki, 2004, Fixed priority scheduling for reducing overall energy on variable voltage processors, in *Proceeding of the 25th IEEE International Real-Time Systems Symposium*, pp. 309-318.

Rao, V., Singhal, G., and Kumar, A., 2004, Real time dynamic voltage scaling for embedded systems, in *Proceedings of 17th International Conference on VLSI Design*. pp. 650-653.

Regehr, John, and Duongsaa, Usit, 2005, Preventing interrupt overload, *Proceedings of the 2005 ACM conference on Languages, compilers, and tools for embedded systems*.

Regehr, John, and et al., 2003, Evolving real-time systems using hierarchical scheduling and concurrency analysis, *Proceedings of the 24th IEEE Real-Time Systems Symposium*, Cancun, Mexico.

Reshadi, M., Mishra, P., and Dutt, N., 2003, Instruction set compiled simulation: A technique for fast and flexible instruction set simulation, in *Proceedings of Design Automation Conference*, pp. 758-763.

Rao, A. and Pande, S., 1999, Storage assignment optimizations to generate compact and efficient code on embedded DSPs, in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 128-138.

Ros, M. and Sutton, P., 2003, Compiler optimization and ordering effects on VLIW code compression, in *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp. 95-103.

Saewong, S., and Rajkumar, R., 2003, Practical voltage-scaling for fixed-priority RT-systems, in *The 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 106.

Saputra, H., Kandemir, M., Vijaykrishnan, N., Irwin, M. J., Hu, J. S., Hsu, C.-H., and Kremer, U., 2002, Energy-conscious compilation based on voltage scaling, in *Proceedings of the 2002 Joint Conference on Languages, Compilers, and Tools for Embedded Systems & Software and Compilers for Embedded Systems*, pp. 2-11.

Shin, Y., and Choi, K., 1999, Power conscious fixed priority scheduling for hard real time systems, in *Annual ACM IEEE Design Automation Conference*, pp. 134-139.

Shin, Y., Choi, K., and Sakurai, T., 2000, Power optimization of real-time embedded systems on variable speed processors, in *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, pp. 365-368.

SimpleScalar simulator, 2004, http://www.simplescalar.com.

Simunic, T., Benini, L., Acquaviva, A., Glynn, P., and Micheli, G. D., 2001, Dynamic voltage scaling and power management for portable systems, in *Proceedings of the 38th Conference on Design Automation*, pp. 524-529.

SourceForge; http://acpi.sourceforge.net/

Srivastava, A. and Wall, D. W., 1994, Link-time optimization of address calculation on a 64-bit architecture, in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 49-60.

Srivastava, M. B., and Chandrakasan, A. P., 1996, Predictive system shutdown and other architecture techniques for energy efficient programmable computation, *IEEE Trans. VLSI Systems*, **4**(1), pp. 42-55.

Swaminathan, V., and Chakrabarty, K., 2002, Pruning-based energy-optimal device scheduling for hard real-time systems, in *Proceedings of the 10th International Symposium on Hardware/Software Codesign*, pp. 175-180

Swaminathan, V. and Chakrabarty, K., 2003, Energy-conscious, deterministic I/O device scheduling in hard real-time systems, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, **22**(7), pp. 813-833.

Swaminathan, V., and Chakrabarty, K., 2005, Pruning-based, energy-optimal, deterministic I/O device scheduling for hard real-time systems, *ACM Trans. on Embedded Computing Systems*, **4**(1), pp. 141-167.

Thuresson, M. and Stenstrom, P., 2005, Evaluation of extended dictionary-based static code compression schemes, in *Proceedings of the 2nd conference on Computing frontiers*, pp. 77-86.

Tian, L., and Arslan, T., 2003, A genetic algorithm for energy efficient device scheduling in real-time systems, in *Proceedings of 2003 conference on Genetic and evolutionary computation*, pp. 1614-1615.

Vaddagir, Srivatsa, and et al., 2004, Power management in Linux-based systems, *Linux Journa*

Wang, Shaojie, Malik S., and Bergamaschi, R.A., 2003, Modeling and integration of peripheral devices in embedded systems, *Design, Automation and Test in Europe Conference and Exhibition*.

Weinberg, Bill, 2004, Porting RTOS device drivers to embedded Linux, *Linux Journal.*

Weissel, A., Beutel, B., and Bellosa, F., 2002, Cooperative I/O—a novel I/O semantics for energy-aware applications, in *Proceedings of 5th Symposium on Operating Systems Design and Implementation.*

Wolfe, A. and Chanin, A., 1992, Executing compressed programs on an embedded RISC architecture, in *Proceedings of the 25th International Symposium on Microarchitecture*, pp. 81-91.

Yao, F., Demers, A., and Shenker, S., 1995, A scheduling model for reduced CPU energy, in *IEEE Annual Foundations of Computer Science*, pp. 374-382.

Yun, H. S., and Kim, J., 2003, On energy-optimal voltage scheduling for fixed-priority hard real-time systems, in *ACM Trans. on Embedded Computing Systems*, 2(3), pp. 393-430.

Zhuang, X., Lau, C., and Pande, S., 2003, Storage assignment optimizations through variable coalescence for embedded processors, in *Proceedings of the ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, pp. 220-231.

Živojnović, V., Tjiang, S., and Meyr, H., 1995, Compiled simulation of programmable DSP architectures, in *Proceedings of IEEE Workshop on VLSI Signal Processing*, pp. 187-196.

Chapter 6

ENERGY MANAGEMENT TECHNIQUES FOR SOC DESIGN

Hiroto Yasuura, Tohru Ishihara, and Masanori Muroyama
System LSI Research Center, Kyushu University, 3-8-33, Momochihama, Sawara-ku, Fukuoka, 814-0001, JAPAN

Abstract: One of the biggest problems in complicated and high-performance SoC design is management of energy and/or power consumption. In this chapter, we present energy management techniques in system design including HW and SW, SoC architecture and logic design. Dynamic power consumption is the major factor of energy consumption in the current CMOS digital circuits. The dynamic power consumption is affected by supply voltage, load capacitance and switching activity. We present approaches to controlling supply voltage, load capacitance and switching activity dynamically and statically in system architecture and algorithm design levels. We also discuss about the memory architecture for reducing power and energy in HW and SW co-design of SoC. In the future CMOS technology, leakage power consumption becomes dominant, because the threshold voltages are scaled as the transistor size shrinks. We summarize the techniques for reducing leakage power in system architecture design. The contents of the chapter include the following issues; (1) power and energy consumptions in SoC design, (2) tradeoff between energy and performance, (3) tradeoff among energy, QoS (i.e., latency and computational precision), reliability, and flexibility (4) techniques for reducing dynamic power consumption, and (5) leakage power reduction techniques

Keywords: Energy consumption, Power consumption, Reliability, Quality of service, HW and SW co-design

1. INTRODUCTION

In past years, the most serious concerns for the VLSI designer were performance, cost, and reliability. Recently, however, this paradigm has shifted. More specifically, reducing power and/or energy consumption has become one of the most important themes in SoC design. The driving factors of the paradigm shift include the following.

- Popularization of portable electronic devices
- Raising demand for reliable and stable computer systems
- Worldwide environmental destruction

One of the biggest factors which motivate the need for low power SoC is the popularization of portable electronics. The typical power consumption for a portable multimedia terminal is around the range of 10-50 [W] when employed chips are not optimized for low-power. Assuming a battery yielding around 65 watt-hours per kilogram is used, the terminal would require unacceptable six kilograms of batteries for ten hours operation between recharges. If we use 500 grams of batteries, the terminal operates only one hour without recharges. Therefore, it is clear that the power consumption has a strong impact on a value of the portable electronic products.

The second need for low power comes from a strong pressure for designers of high-end products to reduce their temperature. In [Black69], Black mentioned that the Mean Time To Failure (MTTF) of aluminum interconnects exponentially decreases as the temperature of a chip increases. Therefore, cooling down the chip temperature is essential for a reliable and stable operation of computer systems. Contemporary performance-optimized microprocessors dissipate as much as 15-50W at 100-200MHz clock rates. The leakage power issue makes this situation worse, because the leakage power increases exponentially as the temperature of the chip increases. In the future, it is expected that a 10 cm^2 microprocessor with 500MHz clock frequency consumes about 300W. The cost for cooling such chips is huge. Consequently, there is a clear advantage to reducing the power consumed in computer systems. Especially for consumer products whose sales are strongly affected by its price, lowering the power is indispensable.

Worldwide environmental destruction drives the strong need for low power electronic devices. Although the power consumption of each electronic device is small (around the range of 10-50W), they are used anywhere and anytime in today's highly information oriented society. Assuming coverage of such electronic devices in the world is 50%, 3.3 billions of people waste 500 billions of watts of power. In addition, rising IT population accelerates this situation. If we reduce the energy consumption of the electronic devices by 10%, we can save 65 mega tons of oil used in gas turbine power plants per a year or can reduce 50 nuclear power units. In 10-20 years from now, we need to come up with innovative solutions which drastically save the energy of the electronic devices with accelerating the growth of IT population.

Recently, many energy reduction techniques at various levels of abstraction, such as at device, circuit, layout, architectural, and software levels are proposed. Regarding the physical design, energy optimization techniques are well studied. However, there is much scope left to study in the system level such as architectural, algorithm, or software level. In this chapter, we present system level energy reduction techniques which might be essential in SoC design.

The rest of the chapter is organized in the following way. In Section 2, we explain mechanisms of power and energy dissipations in CMOS circuits and summarize basic strategy for reducing power and energy consumptions. Section 3 presents techniques for lowering supply voltage statically or dynamically considering several design tradeoffs. Section 4 presents techniques for reducing switching activity without sacrificing quality of services (QoS). In Section 5, we present techniques for reducing the product of switching activity and load capacitance. Section 6 presents strategies for reducing leakage power and shows several examples in detail. Section 7 summarizes techniques for reducing energy consumption by customizing hardware for the target application. Section 8 concludes this chapter.

2. POWER AND ENERGY CONSUMPTIONS IN SOC

The energy consumption of a system, *E*, can be defined as the summation of both spatial and temporal power consumption of circuits [Weste93] as shown in (1) and (2).

$$P = P_{dynamic} + P_{leak} = \sum_{g \in G} SA(g) \cdot CL(g) \cdot V_{DD}(g)^2 + P_{leak}(g) \qquad (2.1)$$

$$E = \int_0^t P dt \qquad (2.2)$$

P: Power consumption of the target system
$P_{dynamic}$: Dynamic power consumption of the target system
P_{leak}: Leakage power consumption of the target system
SA(g): switching activity of gate *g* (expected number of 0–>1 transitions per second)
CL(g): load capacitance of *g*
$V_{DD}(g)$: operation voltage of *g*
t: Execution time of an application program

We treat the energy consumption, *E*, as an objective function to be optimized, because the energy consumption is closely related to the heat and reliability of chips, battery life time of portable devices, and the number of nuclear and gas turbine power stations required. The main approach is detecting a spatial and temporal *hot spot* and reducing the power consumption of the spot. Since the power consumption, *P*, dynamically changes according to the behavior of the software running on a chip and a location of the logic gate on the chip as shown in Figures 1 and 2, both the software and the hardware should be taken into account for reducing the energy consumption of a SoC chip. As one can see from Equations (2.1) and (2.2), we can reduce the energy consumption of the SoC chip by lowering $SA(g)$, $CL(g)$, $V_{DD}(g)$, $P_{leak}(g)$ and t. However, lowering these parameters sometimes causes an increase of the execution time, a degradation of computational quality, system reliability and design flexibility. The key point of the energy reduction in SoC design is considering design tradeoffs among energy consumption, performance, computational quality, system reliability and design flexibility. The goal is minimizing the energy consumption under the constraint of performance, computational quality, system reliability and/or design flexibility.

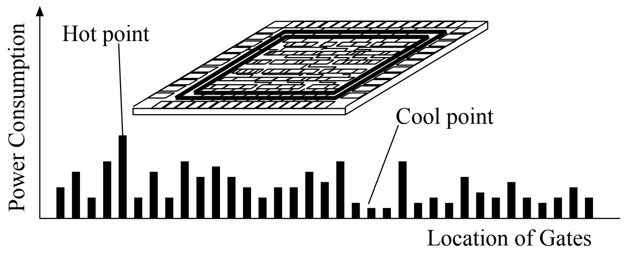


Figure 1. Local Power Dissipation

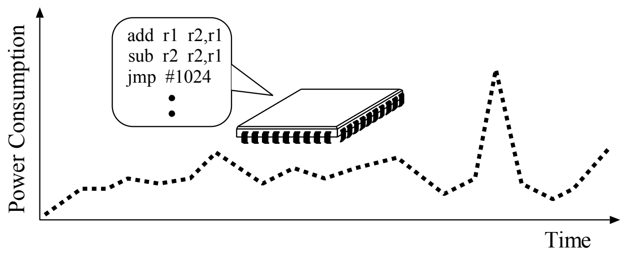


Figure 2. Power Dissipation vs. Energy Dissipation

There is a third source of power consumption, short-circuit power, which results from a short-circuit current-path between the power supply and ground during switching. Short-circuit power is projected to be constant around 10% of total power consumption for succeeding technologies [Chatterjee96]. We ignore it throughout this chapter.

There are many techniques proposed for reducing the execution time *t*, and some of them are very effective for reducing the energy consumption of the SoC chip. In this chapter, however, we do not focus on the techniques which mainly aim to reduce the execution time. Instead, we summarize techniques which consider the execution time as a design constraint. In this chapter, we will make a brief survey on approaches to reducing *SA(g), CL(g)*, $V_{DD}(g)$, and $P_{leak}(g)$ in SoC design. We will also clarify the basic strategy underlying the approaches and show several examples in detail.

3. TECHNIQUES FOR LOWERING OPERATING VOLTAGE

Since energy dissipation is quadratically proportional to supply voltage (see equation (2.1)), lowering the V_{DD} has a strong impact on the energy reduction. However, the following drawbacks should be taken into account;

1. loss of compatibility to external voltage standards,
2. performance degradation, and
3. reliability issues (very low voltage).

3.1 Compatibility of Different Voltage Standards

Whenever one circuit has to drive an input of another circuit operating at a higher supply voltage, a level conversion is needed at the interface. Suppose we have two different voltages, V_{DH} and V_{DL} ($V_{DH} > V_{DL}$). If the output of a circuit operating at V_{DL} is connected directly to the input of a circuit operating at V_{DH}, the static current flows in the input cells operating at V_{DH}, because the PMOS of the input cells cannot be cut-off as shown in Figure 3.

In these days, it is common to have level shifting cells in a cell library for accepting multiple signal levels on a chip. Usami et al. proposed a clustered voltage scaling technique which assumes two different voltages available and finds the optimal voltage assignment to each cell considering the overhead of level shifting cells [Usami95]. Johnson et al. proposed a multiple voltage scheduling technique for reducing the energy consumption of a data path circuit considering an energy overhead of level shifting circuits [Johnson97].

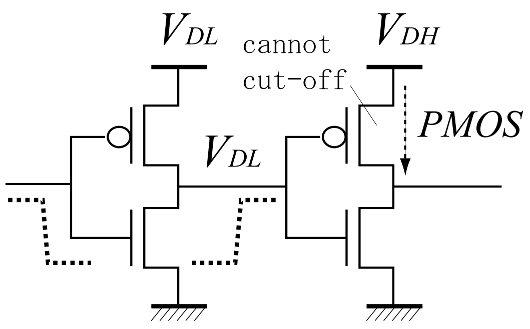


Figure 3. Static Current in Low Voltage Circuits

3.2 Power-Delay Tradeoff

Although lowering the supply voltage is the most effective way for reducing the energy consumption of SoC chip, this causes an increase of circuit delay, τ, which determines the maximum clock frequency of synchronous circuits. The delay τ of a CMOS circuit can be approximately formulated as (3.1),

$$\tau \propto \frac{V_{DD}}{\left(V_{DD}-V_{th}\right)^2} \cong \frac{1}{V_{DD}} \tag{3.1}$$

where V_{th} is the threshold voltage of CMOS transistors used in the circuit.

Basically, we have the following three ways for lowering the operating voltage without sacrificing the performance of the system.

1. Parallelize tasks so that the performance does not degrade even in a low voltage operation. We refer this approach as *static voltage scaling*.
2. Use the maximum available supply voltage for gates on a critical-path and use a lower supply voltage for the other gates. We refer this approach as *multiple voltage assignment*.
3. Lower the clock frequency and operating voltage when the maximum performance is not needed. We refer this approach as *dynamic voltage scaling*.

3.2.1 Static Voltage Scaling

Suppose we have four sequential tasks as shown in Figure 4 (a) and we have two processing units each of which can complete each task per a unit time T_{UNIT} when 5.0V is used. If the tasks can be concurrently run on the processing units as shown in Figure 4 (b), the clock frequency and operating voltage of the processing units can be reduced by half without degradation of system performance. Although switching activities per a unit time may increase up to twice, we can reduce the number of cycles and V_{DD} by half. As a result, energy consumption can be quarter without performance degradation.

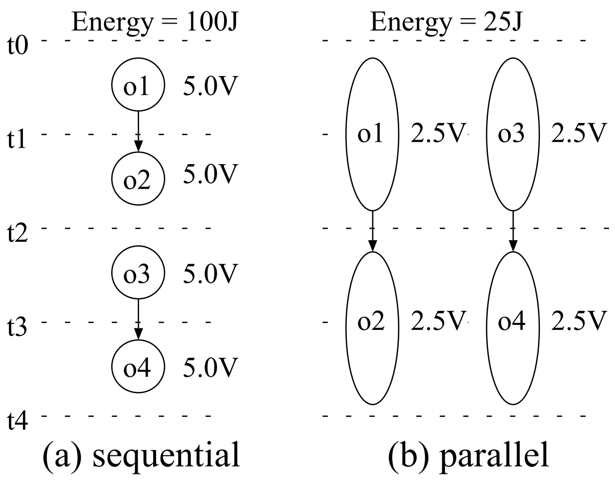


Figure 4. Energy Reduction by Parallel Computation

A lot of researchers have proposed methods that incorporate architectural-level voltage scaling. Chandrakasan et al. proposed HYPER-LP which optimizes dataflow graph generated from a target application program for reducing the power consumption of data-path circuits [Chandrakasan95]. Other methods try to transform the target circuit during scheduling, module selection, resource binding, etc., for minimizing power consumption [Raghunathan94][Raghunathan95][Coodby94][Kumar95][Martin95]. All of the methods mentioned above try to exploit parallelism in the algorithm to shorten critical paths so that lower supply voltage can be used. Although this is a very attractive approach, parallelization of the computation is generally difficult because some computations are inherently sequential.

3.2.2 Multiple Voltage Assignment

Most voltage scaling techniques assume that the circuit operates at a single supply voltage. Although substantial energy savings can be achieved with a single minimum supply voltage, one cannot always take full advantage of available schedule slack to reduce the supply voltage. Since path delays in the circuit are not uniform, supply voltage of gates on a non-critical path can be lowered until the path delay meets with the clock period. When there are nun-uniform path delays, the critical path delay determines the clock period. In this case, non-critical paths use only part of a clock period. The slack time within these clock periods goes to waste. Additional voltages make it possible to use the entire clock period. The basic idea is to assign lower V_{DD}s to the non-critical paths in a way that the delays of the paths meet with the clock period as shown in Figure 5.

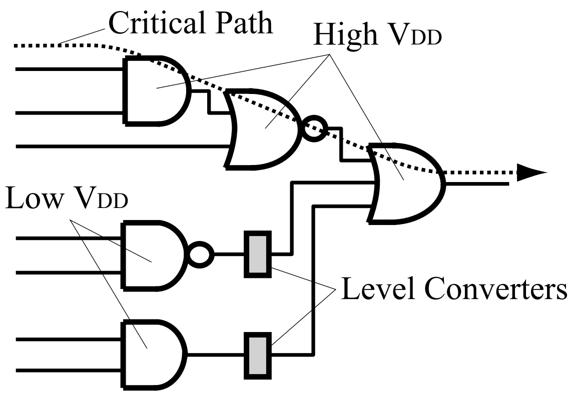
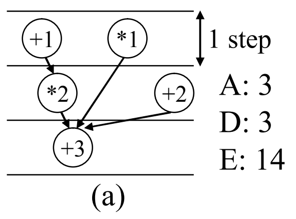


Figure 5. An Example of the Multiple Voltage Assignment

Usami et al. proposed a voltage assignment algorithm which finds the optimal voltage assignment to each cell considering a level shifting cell between different voltages [Usami95]. The algorithm performs backward graph-traversal for a given netlist from the primary outputs toward the primary inputs using the Depth-First-Search (DFS) algorithm. Each time the algorithm visits a cell and tries to replace a high V_{DD} cell with low V_{DD} cell. If the timing constraint is still met even after the replacement, the cell is replaced. This process is repeated until all the cells are visited. Their experiments demonstrated that the energy consumption can be reduced by 20% using two voltages 5V and 3V.

The idea can be extended to a multiple voltage datapath scheduling technique in high level synthesis. The main idea is to minimize energy consumption by assigning operations to time steps with various supply voltages under a given time or resource, or both constraints. We use Figure 6 to illustrate the multiple-voltage scheduling technique. Assume the energy consumption of an addition operation is 1.0 at 1.2V and 2.0 at 1.7V. It requires 2 time steps at 1.2V but only 1 step is sufficient when 1.7V is applied. The area required for the adder module is 1.0. Similarly, the energy consumption of a multiplication operation is 2.0 at 1.2V and 4.0 at 1.7V. It requires 2 time steps and 1 step at 1.2V and 1.7V, respectively. The area required is 2.0. Suppose we have a control flow graph as shown in Figure 6 (a). It needs 3 steps and the energy consumption is 14. Since we can share the resources, we only need one adder circuit and one multiplier circuit in this case. As a result, the area required is 3. Since operations *1 and +2 are not located on a critical path, we can assign lower voltage to them as shown in Figure 6 (d). In this case, energy consumption can be reduced to 11. If we relax the time constraint to 5, we can reduce the energy consumption to 8 as shown in Figure 6 (b). This idea is extended in the following papers [Johnson97][Raje95][Lin97][Chang96] so as to fit with more practical situations.



Module	Area (A)	Delay (D)	Energy (E)
Adder at 1.2V	1.0	2.0	1.0
Multiplier at 1.2V	2.0	2.0	2.0
Adder at 1.7V	1.0	1.0	2.0
Multiplier at 1.7V	2.0	1.0	4.0

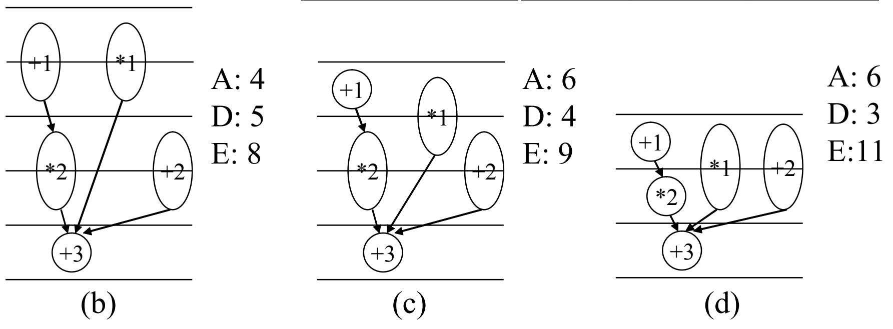


Figure 6. Multiple-Voltage Scheduling in High Level Synthesis

Raje et al. proposed a datapath scheduling technique which schedules the datapath operations, selects voltages from a predetermined set of voltages and assigns the voltages to the datapath operations simultaneously so as to minimize power consumption [Raje95]. Lin et al. used an integer linear programming approach to schedule datapath operations, choose voltages from a list of candidates, and assign voltages to each operation considering timing and resource constraints together [Lin97]. Johnson et al. used an integer linear programming approach to choose voltages from a list of candidates, schedule datapath operations, and assign voltages to each operation considering the energy overhead of level converters [Johnson97]. Chang et al. proposed a dynamic programming approach to optimize nonpipelined datapaths and modified list scheduler to handle functionally pipelined datapaths [Chang96].

3.2.3 Dynamic Voltage Scaling

More aggressive approach is dynamic voltage scaling. Since the computational load is not constant during the execution of given tasks, we can control computational power according to the computational load. The basic idea is assigning different operating voltages to the tasks in a way that any of the tasks does not violate a timing constraint. The assignment can be done statically or dynamically.

Figure 7 shows motivational example of the dynamic voltage scaling. Suppose we have a processor which uses three different supply voltages, 5.0V, 4.0V, and 2.5V. A task which takes 1 billion cycles to complete runs on the processor. The energy consumptions for the task are 10nJ/cycle, 25nJ/cycle and 40nJ/cycle at 2.5V, 4.0V and 5.0V, respectively. The computational speeds of the processor at 5.0V, 4.0V, and 2.5V are 50 million cycles per second, 40 million cycles per second, and 25 million cycles per second, respectively. This assumption follows the Equations (2.1), (2.2) and (3.1).

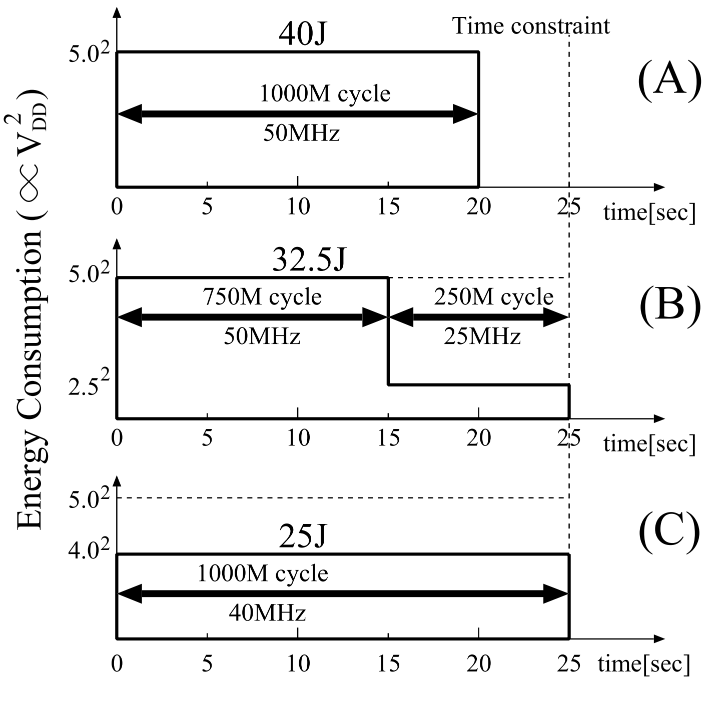


Figure 7. Motivational Example

In Figure 7 (A), the processor uses the maximum supply voltage, 5.0V, for the entire execution of the task. In this case, the total energy consumption is 40J. If the processor uses 2.5V and 5.0V in a way that the completion time of the task meets with a given time constraint, the energy consumption can be reduced to 32.5J as shown in Figure 7 (B). Figure 7 (C) shows the best case of this example. If the processor uses a single supply voltage which adjusts the completion time just to the time constraint, the total energy consumption is minimized.

In [Ishihara98], Ishihara and Yasuura proved the following theorem; if the processor uses a voltage, v_{ideal}, and completes a given task just at a timing constraint T_{const}, the v_{ideal} is the ideal voltage which minimizes energy consumption for the task. The example shown in Figure 7 demonstrates that reducing the energy consumption of the processor is fundamentally equivalent to exploiting idle intervals of the processor. Thus, we should first identify sources of idle intervals to efficiently reduce the power consumption of the processor. There are three major sources as follows;

1. The first one occurs when a system is not tightly designed for a given processor. In other words, there is a room for design change or improvement such as introducing more tasks, replacing certain tasks with their version up, using lower performance processors and so on.
2. The second source comes from a nature of a fixed-priority scheduling. The idle intervals inhere in the fixed-priority scheduling, because the priorities statically assigned to the tasks are not always optimal for the tasks.
3. The third source comes from run-time variation of execution time. Since most of tasks complete its execution much earlier than the worst case execution time, the slack time will be yielded depending on input data for the task.

Consider the three tasks given in Table in Figure 8. T_i, D_i and C_i denote period, deadline and the worst case execution time (WCET) of each task, respectively. Priorities are assigned in row order as shown in the fifth column of the table. Assume all tasks are released simultaneously at time 0. A typical schedule, which assumes that tasks run at their WCETs (C_i), is shown in Figure 8 (a). Note that this system is designed to meet its schedulability. For example, if τ2 takes a little longer to complete, τ3 would miss its deadline at time 100. Even though the system is tightly designed, there are still some idle time intervals, as shown in Figure 8 (a). At time 160 in the figure, when the request for τ2 arrives, the run-time task scheduler knows that there will be no requests for any tasks until time 200, which is the time when requests for τ2 and τ3 will arrive. As a consequence, we can save power by reducing the speed of the processor by lowering the clock frequency and supply voltage. When tasks are completed earlier than their

WCET, we have more chances to apply the same mechanism. For the example of Figure 8 (b), we can slow down the processor at time 50 because the first instances of τ2 and τ3 complete their execution earlier than the second request for τ1 arrives. Since the execution time of each task frequently deviates from its WCET during the operation of the system, we have many chances to slow down the processor as shown in Figure 8.

	Ti	*Di*	*Ci*	*Priority*
τ1	50	50	10	1
τ2	80	80	20	2
τ3	100	100	40	3

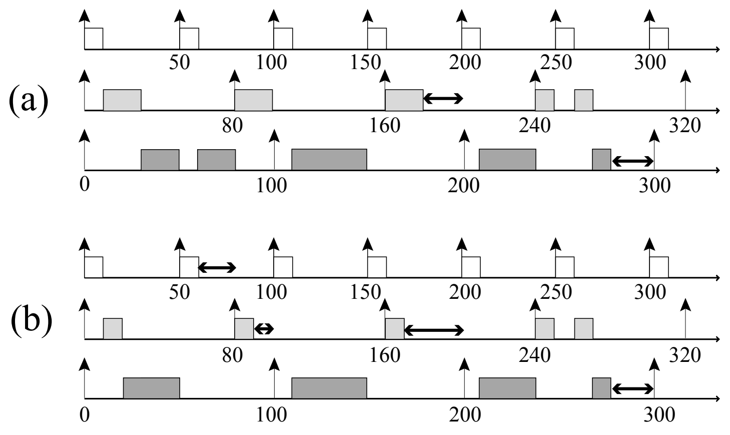


Figure 8. An Example of Task Scheduling on a Variable Voltage Processor

Weiser et al. proposed a scheduling method for dynamically variable voltage processors [Weiser94]. Yao et al. proposed real-time task scheduling methods for the dynamically variable voltage processors [Yao95]. Both of them assume a fixed amount of execution time and exploit the first source of idle intervals only.

In [Shin99], Shin et al. proposed a fixed-priority scheduling method which exploits the second and third sources of idle intervals mentioned above. They extended this work and proposed off-line and on-line algorithms for exploiting all of idle intervals mentioned above [Shin00]. The off-line algorithm finds the lowest possible voltage which guarantees time constraints of all tasks. The on-line algorithm dynamically varies the processor speed along with the supply voltage in order to exploit execution time variations and idle intervals.

In [Okuma99], Okuma et al. proposed a real-time task scheduling algorithms for the dynamically variable voltage processor. Their approach based on the Earliest Deadline First (EDF) algorithm. Similar to [Shin00], their approach exploits the first and third sources of the idle intervals mentioned above. However, they assume to choose voltages from a limited number of candidates, while [Shin00] assumes to use continuous values of voltage and clock frequency which is practically impossible.

3.3 Power-Reliability Tradeoff

Since the voltage scaling technique reduces voltage margins, it is impossible to discuss about low-power design techniques without considering reliability issues. Most circuit designers have to determine supply voltage of the target circuit to ensure that all circuits operate correctly even in the worst-case operating environment. There are three measure voltage margins as follows [Austin04].

1. Process Margin
   This ensures that performance uncertainties resulting from manufacturing variations in transistor do not prevent slower devices from completing computation within a clock period.
2. Ambient Margin
   This ensures correct operation at the worst-case temperature.
3. Noise Margin
   This protects against a variety of noise sources that introduce uncertainty in supply and signal voltage levels, such as di/dt noise in the supply voltage and cross-coupling noise in logic signals.

The sum of these voltages defines the minimum supply voltage that ensures correct circuit operation even in the worst-case condition. As mentioned before, the energy consumption of CMOS circuit is quadratically proportional to the supply voltage. Therefore, it is clear that there is a trade-off between reliability and energy consumption.

Worm et al. proposed an interconnect system which uses low-swing signaling, error detection codes, and a retransmission scheme [Worm02]. This technique optimally finds the interconnect voltage swing and frequency with subject to workload requirements and signal to noise conditions. The most straightforward way to reduce the energy consumption for the communication is lowering the voltage swing of signals propagated through interconnects. This however causes an increase of sensitivity to noise sources because of the decreased noise margins. Their technique monitors bit error rates of the interconnect on the fly as shown in Figure 9 and dynamically finds the optimal swing level which minimizes energy consumption while satisfying the reliability constraint. Their simulation results show that the energy consumption can be reduced by 56% over a conventional interconnect with more robustness to large variations in actual workload, noise and technology quality.

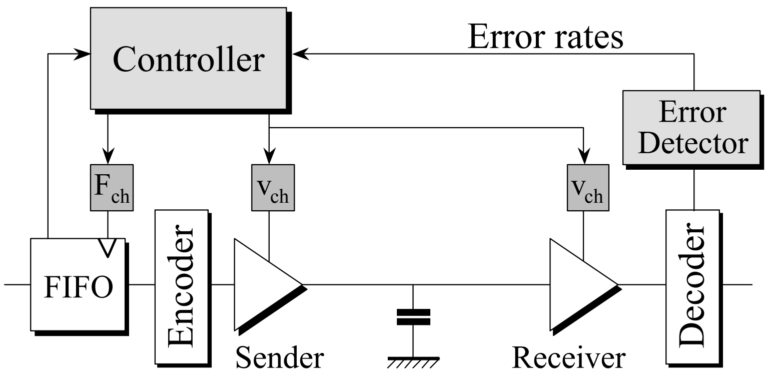


Figure 9. Dynamic Voltage Scaling for Reliable Data Transmission

Bertozzi et al. evaluated energy efficiency of several error resilient techniques such as error correcting codes, a data retransmission technique and so on [Bertozzi02]. Their experiments demonstrated that retransmission strategies are more effective than the error-correction-based technique in terms of energy efficiency.

Austin et al. proposed Razor, a voltage scaling technique based on dynamic detection and correction of circuit timing errors [Austin04]. The technique eliminates unnecessary voltage margins that the traditional worst-case design methodologies require. In some cases, computations may fail and require additional time and energy for recovery. However, the overall computation consumes significantly less energy than traditional worst-case design.

3.4 Commercial Products

There has been a lot of power management software released before. Early power management software used the BIOS to determine whether a device had been idle long enough to shift a sleep state. With the introduction of Advanced Power Management (APM) the OS began to control the power settings and timings. With the Advanced Configuration and Power Interface (ACPI) specification, all power management moved from the BIOS to the hardware and operating system. In today's low-power oriented computer systems, chipsets support ACPI power and thermal management functions to control various system-level and processor-level power and sleep states, and they also still support APM. However, neither APM nor ACPI supports dynamic voltage scaling of chipsets. Recently, many computer systems including laptop PCs, PDAs, cellular phones, and etc. introduced the dynamic voltage scaling technique. The following power management software support dynamic voltage scaling.

- SpeedStep^{TM}, Extended SpeedStep^{TM} (Intel)
- PowerNow! ^{TM} (AMD)
- LongHaul^{TM} (VIA Technologies)
- LongRun^{TM}, LongRun2^{TM} (Transmeta)
- SmartReflex^{TM} (TI)
- IEM^{TM} (ARM)

Most of the above software products are based on the dynamic voltage scaling techniques mentioned in this section. Some of them also support a dynamic body biasing technique which can dynamically control the threshold voltage of transistors for reducing the leakage power consumption of a chip. The detailed explanation of the dynamic body biasing technique will be provided in Section 6.

3.5 Conclusions

In this section we addressed several techniques for lowering supply voltage of chips considering voltage compatibility, a power-delay tradeoff and a power-reliability tradeoff. As mentioned above, lowering supply voltage has the biggest impact on power reduction. The techniques can be applied to many kinds of SoC implementations like multi-chip module (MCM), network on chip (NoC), system in package (SiP), chip multi processor (CMP) and so on. However, it becomes more difficult in future to control supply voltage due to the reliability issues. Breakthrough will appear if we can tolerate negative effects of process variations, temperature variations, soft errors and noises even in ultra low-voltage operation.

4. TECHNIQUES FOR REDUCING SWITCHING ACTIVITY

Lowering the switching activity is a very promising way of decreasing the power consumption. There are numerous researches on this issue. In this section, we introduce system level approaches for reducing the switching activity. System level switching activity reduction can be categorized as follows:

- Turn off unused HW modules.

- Adjust datapath, the bit width of buses and operational units in a system.
- Trade precision for low power (Use narrow bit width).
- Compiler based instruction scheduling.

Practical strategies we pick up in this section are shutting down unused modules, adjusting datapath width to minimize power consumption and compiler optimization techniques for reducing the switching activity.

There are two main shutting down strategies: *clock gating* and *power gating* as summarized in Figure 10. Power gating is mainly used for reducing leakage power. Section 6 describes the power gating in more detail. The best-known technique for reducing the switching activity is clock gating.

	Clock Gating	Power Gating
Hardware support	Easy	Difficult
Overhead (power & area)	Small	Large
Overhead (delay)	Small	Large
Energy efficiency	Moderate	Large

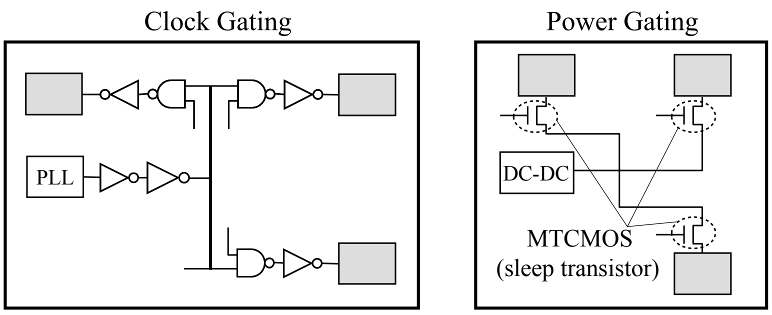


Figure 10. Comparison of Clock Gating and Power Gating

Clock network power can account for as much as 75 percent of the total switching power of a chip, and sequential cells driven by clocks can account for as much as 70 percent of the total clock power. Clock gating essentially disables the clock to a circuit to save power by both preventing unnecessary activity in logic modules and by eliminating power dissipation on clock network. Using a simple AND or OR gate (depending on the edge on which flip-flops are triggered) with the enable and clock signals as inputs, produces a gated clock as output. One can also employ a level-sensitive latch to hold the enable signal from the active edge until the inactive edge of the clock. Clock gating can be applied in either fine-grained or coarse-grained manner.

Fine-grained allows us to reach miscellaneous small units in clock sinks and aggressively save their dynamic power even for a few cycles. Coarse-grained gating saves power from higher level of the clock tree by removing all clock switching from its down-stream units.

Another strategy for reducing switching activity is datapath width adjustment. Since datapath width, the bit width of buses and operational units in a system, strongly affects the size of circuits and memories in a system, the power consumption of a system also depends on the width of the datapath. In design of embedded systems and System-On-a-Chip (SOC), designers have to consider the trade off among system performance, cost and power consumption. Bitwidth of data, the length of data, computed in the system is one of the most important design parameters related with performance, cost and power of the system. The bitwidth of datapath and the size of memories strongly depend on the bitwidth of data. Providing more datapath width for computation than required, will consume more dynamic power and leakage power than necessary by the extra bits.

Typical algorithms defined in C/C++ or SystemC will initially not contain definitions of the actual bit width for operations and storage elements. For algorithm selection, the design team often relies on floating point and straight integer calculations. Based on the stimulus which is applied to the design under optimization users can assess the minimum and maximum values on specific operations and then choose the optimal bit width accordingly. This allows users to understand the impact of bit width on energy and is a step towards trade offs between *quality*, which may be higher in a video application using higher bit width, vs. energy which decreases with lower bit width in the operations. In quality driven design, both higher and lower bits of data can be reduced. From the requirements on the output quality, lower bits of data may be omitted in the datapath width adjustment (See Figure 11). This means that there is potential for further energy reduction by decreasing computation accuracy.

In this section, we describe dynamic power management by using the shutting down strategy, the datapath width adjustment strategy, and instruction scheduling.

4.1 Dynamic Power Management (DPM)

System level dynamic power management (DPM) has gained considerable attention in recent years as a way to save energy in devices that can be turned on and off. DPM dynamically reconfigures systems to provide the requested services and performance levels with a minimum number of active components or a minimum load on such components. The fundamental premise for the applicability of DPM is that systems and their components experience non-uniform workload during operation time and that it is possible predict, with a certain degree of confidence, the fluctuations of workload. There are two power reduction methodologies with idle modes: voltage scaling with frequency scaling and clock gating. Only clock gating methodology is introduced.

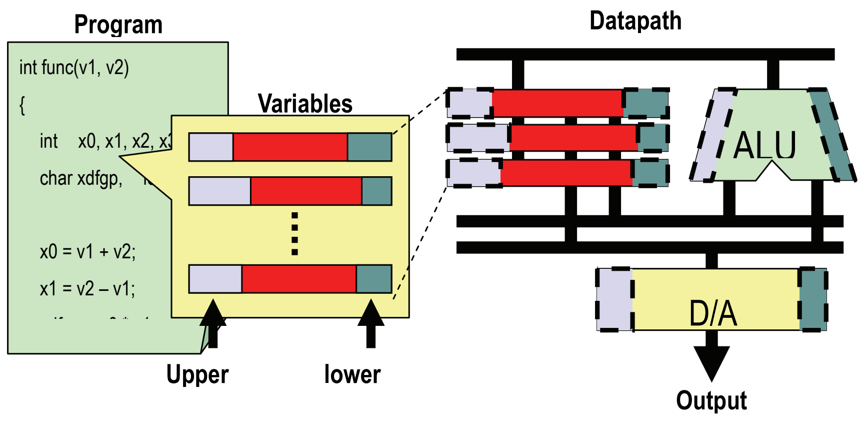


Figure 11. Datapath Adjustment

The control procedure is often called policy. An example of a simple policy, ubiquitously used for laptops and palmtops, is the timeout policy, which shuts down components after a fixed inactivity time, under the assumption that it is highly likely that a component remains idle if it has been idle for the timeout time. Power could be shut off or gated to functional blocks when operating in a standby mode and restored as needed. The gated circuit would not dissipate any power when turned off. Additional circuit would be required to monitor the need for these functional blocks. A problem with power gating is the latency between when the signal to turn a unit on arrives and when the unit is ready to operate. Retention flip-flops on an isolated power supply could be used to save the logic state of all sequential elements when a chip is powered down, eliminating the need to reinitialize the device when it comes out of standby mode. Some products support multiple levels of standby (soft off, nap and sleep) which differ in terms of the amount of power saving and latency (See Figure 12).

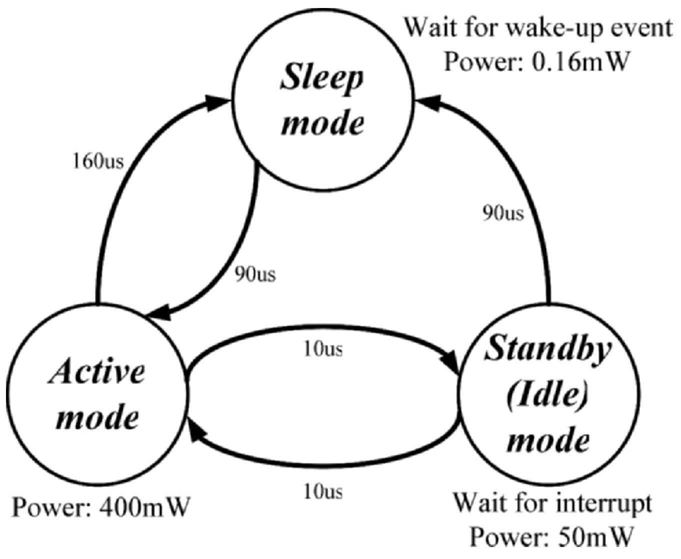


Figure 12. Dynamic Power Managament

4.2 Datapath Width Adjustment (Bit-width optimization)

Processor-based systems treat various data with different bit width. It is efficient in power reduction not only to determine datapath width statically but also to control the active datapath width dynamically.

First, we introduce static optimization, which adjusts datapath width. Bit-width analysis is performed to extract information on the required bit width of variables in programs and algorithms. For hardware design, using the result of bit-width analysis, one can determine the length of registers, the size of operation units, and the width of memory words on the datapath of a system to minimize the meaningless power consumption by the useless bits. Shorter registers and operation units reduce switching activity and the leakage of extra bits on the datapath. However, the trade-off between power consumption and execution time needs to be resolved. Generally, narrowing the datapath width reduces the area and power of the processor, but degrades the performance. The number of execution cycles increases, since some single-precision operations should be replaced with double or more precision operations in order to preserve the accuracy of the computation. Single-precision operation are those whose precision is smaller than that of the datapath width. For example, an addition of two 32-bit data is a single-precision operation whose datapath width is equal to or greater than 32 bits, while it is a double precision operation on 16-bit processors. Changing the datapath width affects the size of data memory (RAM) and instruction memory, which is mostly implemented by ROM in embedded systems. Let us consider a program including two variables *x* and *y*, and assume that two variables *x* and *y* require at most 18 bits and 26 bits, respectively (see Figure 13). When the datapath width is 32 bits, two words are required to store these two variables, and the amount of the data memory is 64 bits. Since the minimum bit size required to store the variables is only 44 bits (18+26), 20 bits of the memory (about 30%) are unused. By reducing the datapath width to 26 bits, one can reduce the unused bits to 8 bits. Unused bits, however, increase to 31 bits, if a 25-bit datapath is adopted, because *y* requires two words. When the datapath width is 9 bits, two words and three words are required for *x* and *y*, respectively, and the unused area is only 1 bit. Many unused bits in the data memory can be eliminated by datapath-width optimization.

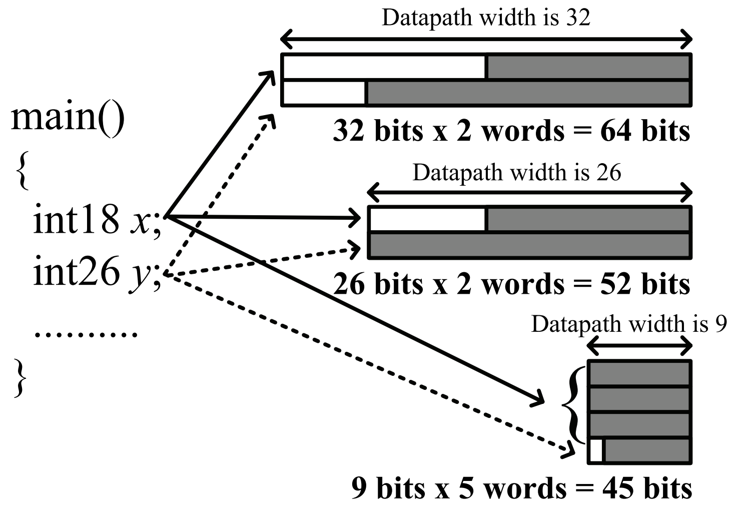


Figure 13. An Example of Datapath Width Adjustment

Second, dynamic approach, which controls active datapath width, is introduced. This approach is called *value-based clock gating*. There is a fact that “narrow-width” data is common not only in multimedia codes, but also in more general workloads. For example, over half the integer operation executions require 16 bits or less on a 64-bit processor. Basic mechanism to reduce power consumption is operand-value-based clock gating to turn off portions of memories, buses, and arithmetic units that will be unused by narrow-width operations. This optimization results in around 50% reductions in the data bus and integer unit power consumption. By applying this for data memory, 80% power reduction can be achieved. However, this approach requires hardware cost for detecting dynamically operation widths and turning off the unused units. As shown in Figure 14, if there is a 7-bit width data, only the lower data memory (D0) is accessed.

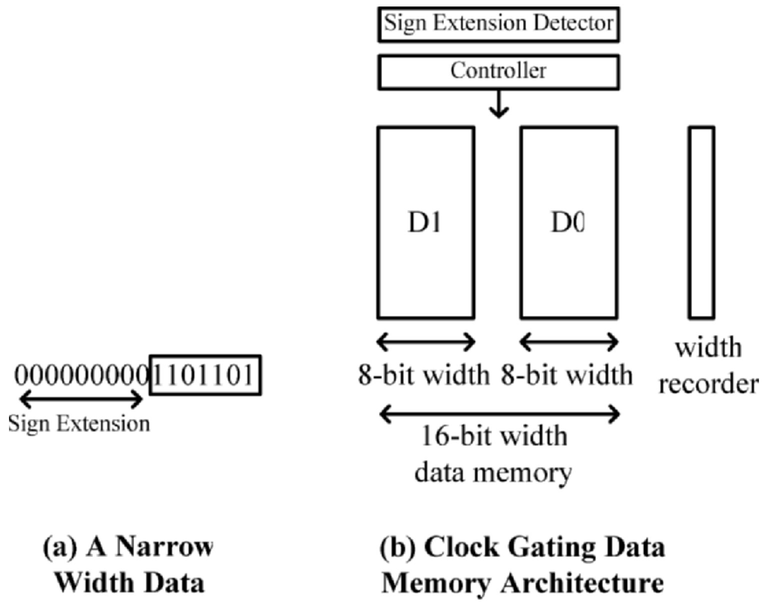


Figure 14. A Data Memory Example Using Operand Based Clock Gating

4.3 Compiler Optimization

Compiler optimization is also effective for reducing the switching. In [Tomiyama1998], they proposed an instruction scheduling technique to reduce power consumption due to off-chip driving. Their technique reduces transitions on a data bus between an on-chip cache and a main memory, and as a result, power consumed by off-chip drives in the main memory, and is reduced. Let us consider an example in Figure 15, and assume 8-bit instruction width and 32-bit cache line size. There are four instructions (a)-(d) in the memory block. When the memory block is sent to the cache, the instruction (a) is sent first. At the time, four bits switch from high- to low-level. At the next cycle, (b) is sent to the cache and six bits switch to opposite level. As a result, the cache miss invokes twenty four transitions totally in the data bus. If changing the positions of two instructions (b) and (c) keeps the meaning of the program, it reduces bus transitions by 25%, from twenty four to eighteen bus transitions (See Figure 15). Thus the instruction scheduling can reduce transitions on the bus. Tomiyama et al. reported that the scheduling algorithm achieves significant reduction in transitions on the data bus, up to 28% of reduction, and runs efficiently.

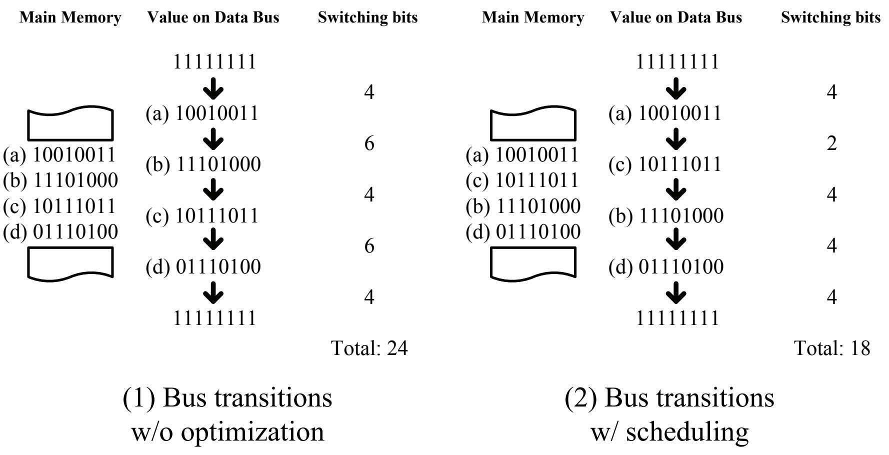


Figure 15. An Example of Instruction Scheduling for Low Power

4.4 Commercial Products

The Pentium 4 processor uses the clock gating technology. Every unit on the chip has a power reduction plan, and almost every functional unit block contains clock gating logic.

4.5 Conclusions

In this section, we summarized system level switching activity reduction strategies. The basic strategies are clock gating and datapath width adjustment. Analyzing statically and dynamically system requirements, unnecessary switching activity reduction can be achieved.

5. TECHNIQUES FOR REDUCING THE PRODUCT OF SWITCHING ACTIVITY AND A LOAD CAPACITANCE

A major contributor to the system budget is the memory-processor interface. Ko et al. mentioned that the power dissipation of an external memory access is at least an order of magnitude higher than that of an on-chip access [Liu94][Ko98]. For this reason, a lot of techniques for reducing energy consumption of the off-chip buses have been proposed. The basic idea is reducing the switching activities (*SA*) of hardware modules whose load capacitance (*CL*) is large even if the *SA*s of low-*CL* modules are increased. Suppose we have a processor system including a CPU core, cache memories, an off-chip memory, and a processor-memory interface as shown in Figure 16. The energy dissipation of the memory-processor interface, $E_{interface}$, can be expressed by (5.1),

$$E_{interface} = N \cdot \left(E_{address} + E_{data} + E_{memory} + E_{overhead}\right) \tag{5.1}$$

where N, $E_{address}$, E_{data}, E_{memory}, and $E_{overhead}$, represent the number of memory accesses, the energy dissipation in address buses per access, that in data buses per access, that in a memory module per access and energy overhead per access, respectively. There may exist the energy overhead if the memory-processor interface is modified for reducing the energy consumption in off-chip buses. As one can see, we can reduce the energy dissipation of the processor-memory interface by decreasing N, $E_{address}$, E_{data}, E_{memory}, and $E_{overhead}$. The problem of minimizing the total energy consumption of the processor system is basically equivalent to finding the best tradeoff point between on-chip computational energy and off-chip communication energy.

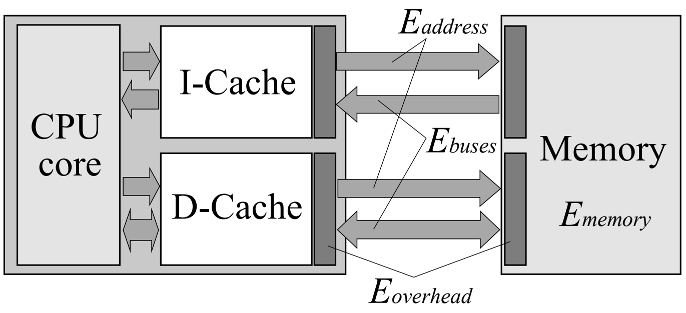


Figure 16. Energy Dissipation of Processor-Memory Interface

There are the following three major approaches for reducing the energy required for the communication between a memory and a processor.

- Cache miss reduction
- Bus encoding
- Code compression

5.1 Cache Miss Reduction

Since cache miss rate is associated with the number of off-chip memory access, reducing cache miss rate leads to a reduction of the energy dissipation for the off-chip memory accesses. The most straightforward way for reducing the cache miss rate is to employ larger cache memory on a chip. Many techniques have been proposed for optimizing cache configuration considering tradeoff between energy consumption of off-chip memory and cache memory [Su95][Hicks97][Li98][Shine99][Malik00]. All these techniques are based on the fact that while a bigger cache consumes more energy per access, it can reduce the number of cache misses and as a result can reduce the energy consumption for the off-chip accesses. Suppose we have a processor with on-chip cache memory which can be resized for the target application as shown in Figure 17.

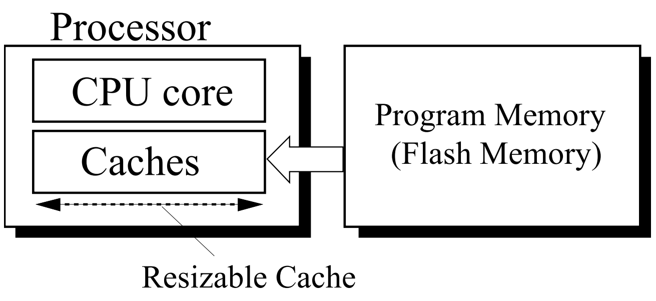


Figure 17. An Example of Resizable Cache

If we optimize the cache size for the target application, the energy consumption for memory accesses can be drastically reduced. For example, based on the experiment in [Ishihara05], the optimal cache size for the SPEC95 benchmark program, "Compress", is 2kB as shown in Figure 18. If we use the 4kB cache instead of 2kB power consumption of the cache becomes very large. Conversely, if the 1kB cache is used, the power consumption of off-chip memory becomes huge due to the large number of cache misses. In the optimal case, the power consumption can be reduced by 85% compared to the result for 1kB cache memory. Note that the leakage power of the cache memory is assumed to be 10% of its dynamic power consumption.

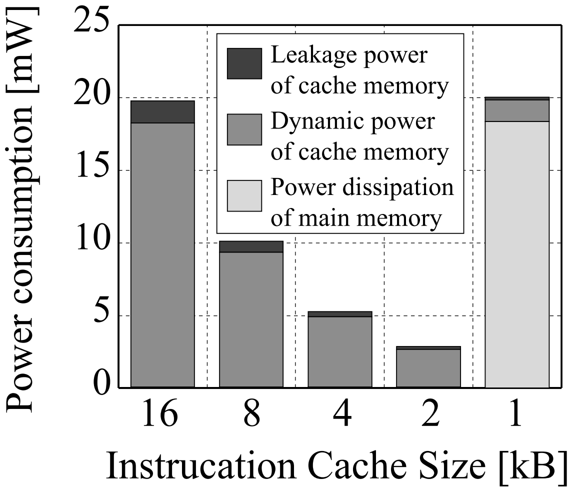


Figure 18. Cache Optimization for Low Power

Li and Henkel proposed *Avalanche* framework which simultaneously evaluates the tradeoffs of energy dissipations of caches and main memory [Li98]. The trade-off between system performance and energy dissipation is also explored in the framework. Their experiments demonstrated significant improvements (up to 95% energy saving) in energy dissipation.

Another approach to reducing the number of cache misses is a compiler-based approach [McFarling89][Hwu89][Tomiyama96][Panda96] [Hashemi97] [Ghosh99]. The idea is to modify the place of basic blocks, procedures, or global variables in the address space so that the number of cache conflict misses is minimized. This can significantly reduce the number of cache misses and energy consumption of memory subsystems. We first explain the idea behind the typical code and data placement technique. Consider a direct-mapped cache of size C (= 2^m words) whose cache line size is L words, i.e., L consecutive words are fetched from the main memory on a cache read miss. In a direct-mapped cache, the cache line containing a word located at memory address M can be calculated by ($\lfloor M/L \rfloor$ mod C/L). Therefore, two memory locations M_i and M_j will be mapped onto the same cache line if the following condition holds,

$$\left(\left\lfloor \frac{M_i}{L} \right\rfloor - \left\lfloor \frac{M_j}{L} \right\rfloor \right) \bmod \frac{C}{L} = 0 \tag{5.1}$$

Several code and data placement techniques have used the above formula [5.6-5.13]. Assume a direct mapped instruction cache with 4 cache-lines, where each cache-line is 32 bytes as shown in Figure 19. Functions A, B, C and D are placed in the main memory as shown in the left side of Figure 19. If functions A, B, and D are accessed in a loop, conflict misses occur because A and D are mapped onto the same cache line. If the locations of C and D are swapped as shown in the right side of Figure 19, the cache conflict is resolved. Code placement techniques modify the placement of basic blocks or functions in the address space so that the total number of cache conflict misses is minimized. Similar to the code placement techniques, data placement techniques modify the placement of global variables in the address space so as to reduce the number of data cache misses.

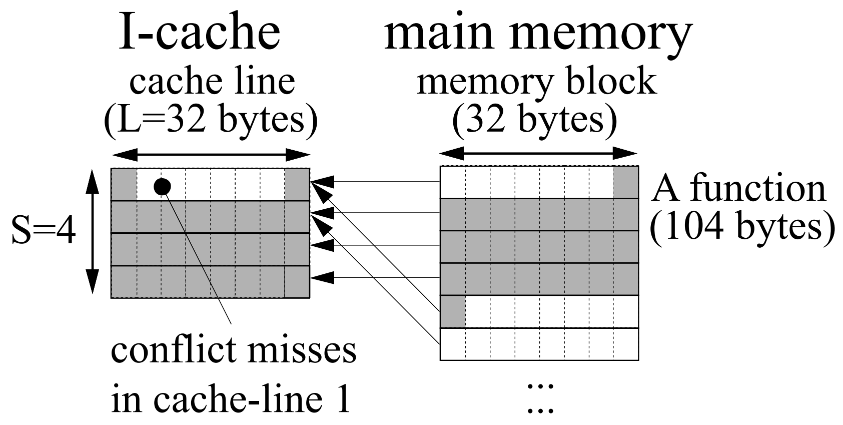


Figure 19. An Example of Code Placement

Kulkarni et al. proposed a data placement algorithm which finds the optimal locations of global variables in the main memory [Kulkarni01]. The algorithm also explores different cache sizes considering trade-offs among performance, energy consumption and chip area. In the first step, they measure the cache miss rates for different cache sizes. Once the miss rates are obtained, the algorithm performs data placement for each cache size and estimate the energy consumption including energies for on-chip accesses and off-chip accesses. Depending on the design constraints, the designer can either choose a lower power solution with some overhead in size and vice versa. Their experiments demonstrated that the total energy consumption can be reduced by 10.6% with 26% performance overhead and 7% area overhead.

Scratchpad memory can be used as a design alternative for the on-chip cache memory. Current embedded processors particularly in the area of multimedia applications and graphic controllers have on-chip scratchpad memories. In cache memory systems, the mapping of program elements is done during runtime, while in scratchpad memory systems this is done by the programmer or the compiler. Unlike the cache memory, the scratchpad memory does not need tag search operations and, as a result, it is more power efficient than the cache memory if programmers or compilers can optimally allocate code and data on the scratchpad memory.

Ishihara and Yasuura proposed a code allocation technique which finds a size of an on-chip scratchpad memory and a code allocation to the scratchpad memory simultaneously so as to minimize the total energy required for fetching instructions [Ishihara00]. Their experiments showed that the energy consumption for the instruction fetching can be reduced by 50%. Benini et al. presented a novel solution for the design hierarchy of low-power embedded systems [Benini00]. The idea is mapping the most frequently accessed data onto a small memory, called application-specific memory (ASM) which is placed vary close to the processor. The experimental results on a set of typical embedded programs have shown that the energy consumption can be reduced by 68% with respect to equivalent caches having different sizes, organizations and configurations. Banakar et al. proposed an approach for selection of on-chip memory configuration from various sizes of cache and scratch pad memories [Bankar02]. Their experiments show that scratchpad based compile-time memory outperforms cache-based run-time memory on almost all aspects. For example, the total energy consumption of scratchpad based systems is less than that of cache-based systems by 40% on an average.

5.2 Bus Encoding

Bus encoding techniques reduce communication power by changing the format of the information in a way that the total communication power is minimized. The basic strategy is to reduce switching activity of off-chip buses by encoding data transmitted between a processor and a memory. We have to consider a tradeoff between the energy consumed in buses and the energy overhead of encoding and decoding circuits. Suppose we have an original data format, *Format-A*, and low-switching format, *Format-B* as shown in Figure 20. Energy consumption for sending data using *Format-A* and *Format-B* is *EA* and *EB*, respectively. The energy overhead for encoding and decoding (i.e., translating *Format-A* into *Format-B* and vice versa) is $E_{overhead}$. Bus encoding techniques are effective only when the following inequality holds,

$$EA > EB + E_{overhead}.$$

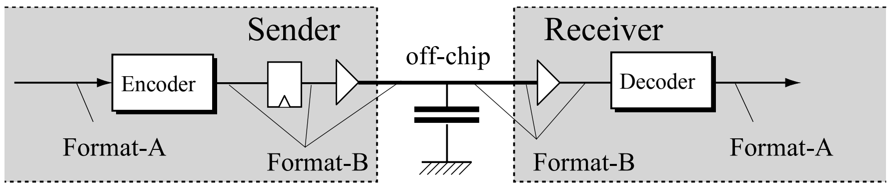


Figure 20. Low-Power Bus Encoding

The bus-invert coding is one of the most popular approaches [Stan95]. In the bus-invert coding, if the Hamming distance (the number of switched bits) between the new pattern to be transferred and the old one currently on the bus is larger than half the bus width, the new pattern is transferred with each bit inverted. An additional invert bit is used to inform the receiver side whether the pattern is inverted or not. The experiments demonstrated that the bus-invert coding technique decreases the I/O peak power dissipation by 50% and the I/O average power dissipation by 25%.

For instruction address patterns, where consecutive patterns are often sequential, the Cray code is efficient [Su94]. The Gray code has only one-bit difference in consecutive number for addressing. Due to locality of program reference, Gray code addressing can significantly reduce the number of bit switches. The experimental results showed that for typical programs running on a RISC microprocessor, using Gray code addressing reduce the switching activity at the address lines by 30-50% compared to conventional binary code addressing.

In the *T0 code* [Benini97], the bus transitions are further reduced by freezing the address lines when consecutive patterns are detected to be sequential. An extra bus line is employed to inform the receiver side whether or not the current pattern is sequential.

In special purpose applications, where the information about the sequence of patterns available a priori, the characteristics of patterns can be exploited to efficiently reduce bus transitions. The Beach Solution [Benini97-2] makes clusters of bus lines based on statistical information of address patterns and then generates an encoding function for each cluster such that the encoded version of each cluster results in less transitions.

For data address patterns which are less sequential than instruction address patterns and less random than data patterns, the Partial Bus-Invert code [Shin98] performs better. It applies the bus-invert coding to a pre-defined sub-group of bus lines thereby avoiding unnecessary inversion of relatively inactive and/or uncorrelated bus lines. The experiments on benchmark examples indicate that the partial bus-invert coding reduces the total bus transitions by 62.6% on the average, compared to that of the unencoded patterns.

5.3 Code Compression

An alternative approach to bus encoding is code compression. The basic strategy is to use narrow instruction codes for reducing the switching activity when the instructions are transmitted from a program memory to a CPU.

One of the best known instruction compression approaches is the "Thumb" instruction set of the ARM microprocessor family [Segars95]. ARM cores can be programmed using a reduced set of 16-bit instructions instead of standard 32-bit RISC instructions, which reduces required instruction memory occupation and bandwidth by a factor of 2.

Yoshida et al. proposed a code compression technique as depicted in Figure 21 [Yoshida97]. Suppose we have an object code and the number of distinct instructions appeared in the code is N. In this case, we can express all those instruction codes using $\lceil \log N \rceil$-bit binary patterns. Since the firmware running on a given embedded processor normally uses only a small subset of the instructions supported by the processor, a $\lceil \log N \rceil$-bit is much smaller than original instruction width. As a result, we can reduce the energy consumption for fetching instruction. According to this idea, the object code is stored in memory in compressed format, i.e., each instruction is replaced with a $\lceil \log N \rceil$-bit binary pattern which is in one-to-one correspondence with the original instruction. Every time an instruction is fetched from the program memory, it is decompressed (i.e., the original format is restored) using an *instruction decompression table* (IDT) and then passed to the processor's decoding logic. This architecture is motivated by the fact that software programs normally use only a subset of all possible instructions offered by the processor's instruction set. Since $\lceil \log N \rceil$ (where N is the number of distinct instructions) is usually much smaller than the original instruction width, this approach reduces both memory energy and bus power consumption.

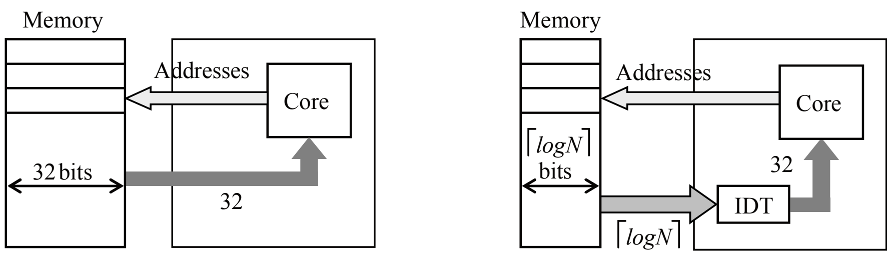


Figure 21. An Example of Code Compression

Although, in principle, the solution depicted above offers good opportunities for energy reduction, it often happens that the number of distinct instructions, *N*, used by a program is not small. In such a situation, the size of the *Instruction Decompression Table* (IDT) becomes very large, and therefore area and power dissipation of the IDT would be very large as well. As a solution of the problem, Benini et al. proposed a selective instruction compressing technique [Benini99]. Their idea is to compress only a subset of fixed cardinality (256 elements) of the instructions used by a program, namely, those that are executed more often. This approach is motivated by the observation that the 256 most frequently used instructions are always executed for at least 50% and up to 99.99% of the time. The idea can be implemented as shown in Figure 22. This approach guarantees a fixed and limited size for the IDT and reduces energy and area overhead for decompressing the instructions.

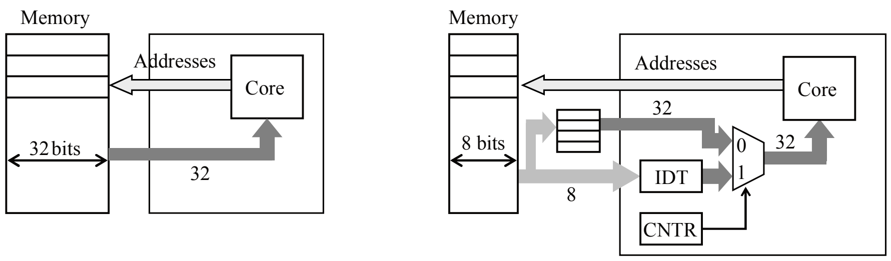


Figure 22. Selective Code Compression

5.4 Conclusions

We addressed several techniques for lowering switching activity of off-chip buses considering tradeoff between the power consumption for on-chip computation and that for off-chip communication. Other than the techniques addressed in this section, there have been proposed a lot of techniques which reduce switching activity of high capacitance nodes. Specifically, circuit level approaches like logic synthesis techniques, placing and routing techniques, and high-level synthesis techniques which reduce transitions of high capacitance modules are well studied. On the other hand, there is much scope left to study on source-level design techniques which modify an application program in a way that power-hungry hardware components are less frequently used without sacrificing performance, computational quality and system reliability.

6. TECHNIQUES FOR REDUCING REAKAGE POWER

For mobile/portable devices with a high standby-to-active ratio, leakage current may be the dominant factor in determining overall battery life. The three primary sources of leakage current (See Figure 23) are sub-threshold (I_{sub}) or source-to-drain leakage current which grows exponential with

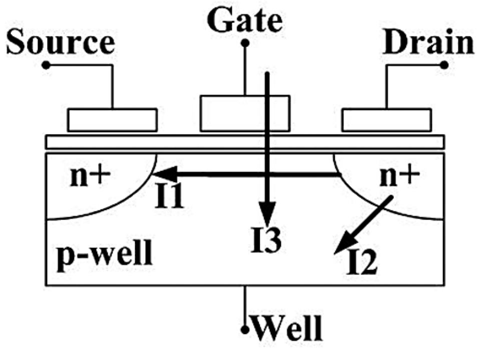


Figure 23. Sources of Leakage Current

lowering V_t and increasing temperature, reverse bias junction band-to-band tunneling current ($I_{b\text{-}b}$), and gate oxide tunneling current (I_{gate}). Reducing of gate oxide thickness results in an increase in the field across the oxide. The high electric field coupled with low oxide thickness results in tunneling of electrons from substrate to gate and also from gate to substrate through the gate oxide, resulting in the gate oxide tunneling current. Most of the interests have focused on the leakage caused by sub-threshold current and gate oxide tunneling current in terms of system level leakage management. Due to the leakage mechanisms described above, leakage current increases dramatically in the scaled devices. Particularly, with reduction of threshold voltage to achieve high performance, leakage power becomes a significant component of the total power consumption in both active and standby modes of operation. Since in the sleep mode I_{gate} will likely be dominant, two approaches may be considered: (1) reduce the threshold voltage of the sleep device somewhat (e.g. 100mV) to minimize the delay penalty associated with an extra series device; this allows the use of smaller sleep devices to simultaneously reduce I_{gate}, dynamic power, and layout area while not penalizing standby mode leakage since $I_{sub} << I_{gate}$ or (2) incorporate a multi-T_{ox} process was proposed.

A key difference between the state dependence of I_{sub} and I_{gate} is that the magnitude of I_{sub} primarily depends of the number of on vs. off transistors in a stack, while I_{gate} also depends strongly on the position of the on/off transistors.

Leakage power can be expressed as follows [6-2]:

$$P_{leak} = n \cdot I_{leak} \cdot V_{DD},\ I_{leak} \propto (V_{TH} / \alpha V_T)(1 - e^{-V_{DD}/V_{TH}}) \quad (6\text{-}1)$$

where n indicates the number of transistors, V_T denotes thermal voltage which is about 25mV at room temperature and increases linearly as temperature increases. According to this relationship, leakage current and therefore power dissipation increases exponentially with decreasing threshold voltage (V_{TH}) and with increasing temperature. Equation (6-1) suggests two ways to reduce P_{leak}. First, we could turn off the supply voltage. That is, set V_{DD} to zero so that the factor in parentheses also becomes zero. Second, we could increase the threshold voltage, which (because it appears as a negative exponent) can have a dramatic effect in even small increments. Of course using high-V_T transistors will degrade performance. A solution is to have mixture of high and low V_T transistors. Use low V_T transistors on timing-critical paths and high Vt transistors on non-critical paths. This approach is referred to as dual V_T design. Multi-Threshold CMOS (MTCMOS) cells can be used to control leakage power (See Figure 24). Low V_T transistors are used to implement gates for high speed, while high V_T transistors are added to form virtual rails. These high V_T transistors suppress the leakage current when the sleep signal is activated. Of course, there needs to be a sleep control mechanism.

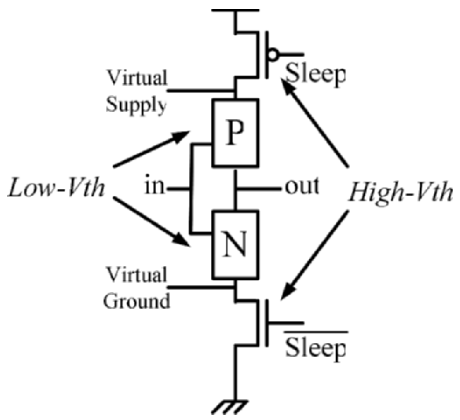


Figure 24. Multi-Threshold CMOS (MTCMOS)

Variable Threshold CMOS (VTCMOS) is a body biasing technique that controls effective threshold voltage by applying substrate bias to MOS transistors (See Figure 25). This technique is applicable at runtime. In the active mode, a zero body bias is applied. In standby mode, the effective threshold voltage is made to be larger by applying a reverse substrate bias to block the leakage current. Transistor performance in the active mode is kept the same as that in the conventional design by utilizing low V_{DD} and low V_T . However, triple well technology is required.

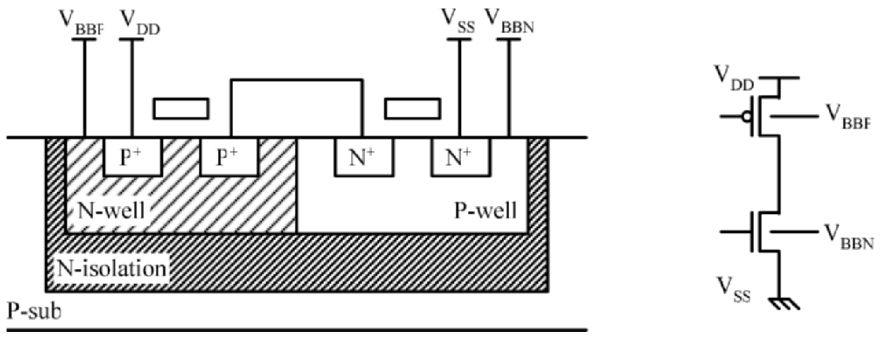


Figure 25. Variable-Threshold CMOS (VTCMOS)

In addition to above approaches, area reduction also reduces leakage power. Datapath width adjustment described in Section 6.4 is also effective for reducing the leakage power. The power dissipation of the whole system not only dynamic power but also leakage power is drastically reduced by tuning the parameters of processors and memories tailored for the applications.

Reducing the number of transistors and controlling power supply voltage, V_T , or temperature dynamically can reduce the leakage. Basic strategies are shown below. Some system level methodologies related using the strategies are shown in this section.

- using high threshold voltage for non-critical paths
- shifting the circuit to the low leakage mode
- cooling high temperature parts,
- reducing the number of transistors.

Many techniques [Ishihara2002, Kaxiras2000, Powell2000, Sato2004] proposed to address leakage power have focused on cache memory that is a major leakage consumer of the entire system because leakage power is a function of the number of transistors. For example, StrongARM processor uses 60% of the die area for cache memories [Manne1998].

6.1 Multiple Vth CMOS and Dual Vth Techniques

One way to increase the threshold voltage is to use Multiple Threshold Circuits with sleep transistors [Calhoun2003]. This involves isolating a leaky circuit element by connecting it to a pair of virtual power supplies that are linked to its actual power supplies through sleep transistors (Figure 24). When the circuit is active, the sleep transistors are activated, connecting the circuit to its power supplies. However, when the circuit is inactive, the sleep transistors are deactivated, thus disconnecting the circuit from its power supplies. In this inactive state, almost no leakage passes through the circuit because the sleep transistors have high threshold voltages. This technique effectively confines the leakage to one part of the circuit, but is tricky to implement for several reasons. The sleep transistors must be sized properly to minimize the overhead of activating them. They cannot be turned on and off too frequently. Moreover, this technique does not readily apply to memories, because memories lose data when their power supplies are cut.

Another way to increase the threshold is to employ dual threshold circuits. Dual threshold circuits [Liu2004, Wei1998, Ho2004] reduce leakage by using high threshold (low leakage) transistors on non-critical paths and leakage by using low threshold transistors on critical paths, the idea being that non-critical paths can execute instructions more slowly without impairing performance.

6.2 Dynamic Power Management for Reducing Leakage

Adaptive body biasing technique [Seta1995, Kobayashi1994,Nose2002] is a runtime technique that reduces leakage power by dynamically adjusting the threshold voltages of circuits depending on whether the circuits are active. When a circuit is not active, the technique increases its threshold voltage, thus saving leakage power exponentially, although at the expense of a delay in circuit operation. When the circuit is active, the technique decreases the threshold voltage to avoid slowing it down. To adjust the threshold voltage, adaptive body biasing applies a voltage to the transistor’s body known as a body bias voltage (Figure 25). Vt is dynamically controlled through software depending on the workload of a processor. The Vth-hopping scheme [Nose2002] can achieve 82% power saving compared with the fixed low-Vth circuits. In order to efficiently suppress the leakage power, combining the adaptive body biasing technique and the dual Vt technique could be useful (See Figure 26). In this case, the adaptive body biasing is used only in the critical paths. On the other hand, Vt of the non-critical paths gates is set to a considerably higher value (high-Vt), which is not changed for the entire time.

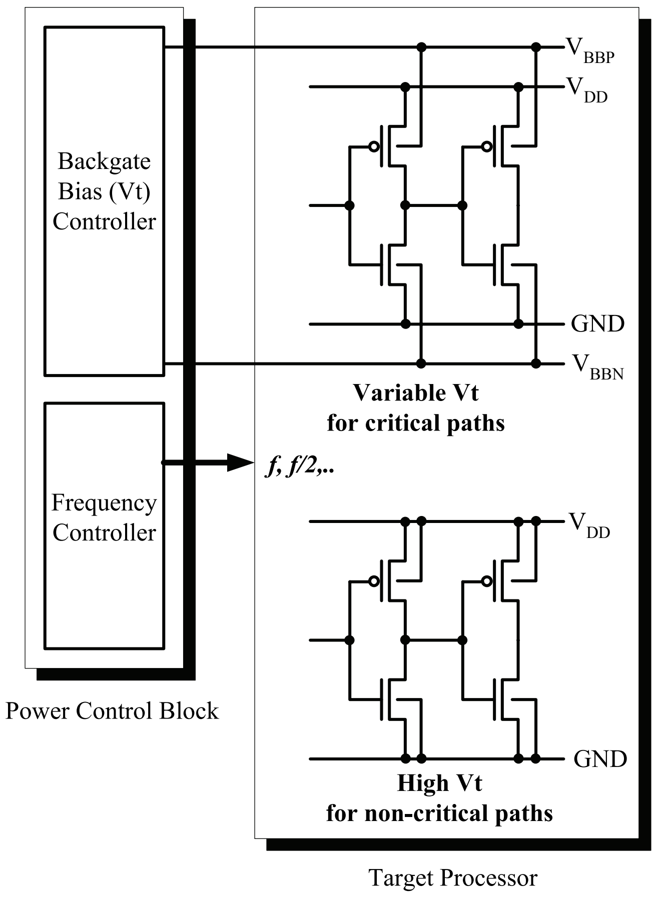


Figure 26. Combining VTCMOS and Dual Vth Technologies

6.3 Thermal Management

Several cooling techniques have been developed since the 1960s. Some below cold air into the circuit, while others refrigerate the processor [Schmidt2002], sometimes even by costly means such as circulating cryogenic fluids like liquid nitrogen [Krane1988]. These techniques have three advantages. First, they significantly reduce subthreshold leakage. In fact, a recent study [Schmidt2002] showed that cooling a memory cell by 50 degrees Celsius reduces the leakage power by five times. Second, these techniques allow a circuit to work faster because electricity encounters less resistance at lower temperatures. Third, cooling eliminates some negative effects of high temperatures, namely the degradation of a chip's reliability and life expectancy. Recently, the reliability is a much more significant issue in design. Despite these advantages, there are issues to consider, such as the costs of the hardware used to cool the circuit. Moreover, cooling techniques are insufficient if they result in wide temperature variations in different parts of a circuit. Rather, one needs to prevent “hotspots” by distributing heat evenly throughout a chip.

Reliability and leakage power are both strongly affected by system temperature. In [Simunic], they proposed a joint reliability and power management optimization. Their approach achieved a significant improvement in energy consumption (40%) in tandem with meeting reliability constraint for all operating temperatures.

Another thermal management is a temperature aware task scheduling [Hung2005], which is task scheduling such that the temperature of HW is minimized.

6.4 Bitwidth Optimization for Reducing Leakage

Cao et. al. [Cao2002] reported a bitwidth optimization technique for reducing not only dynamic and leakage power at system level design. For Lempel-Ziv algorithm, they got dynamic power saving of 59.2% and leakage power saving of 64.3 at the optimal datapath width of 15bits; for ADPCM encoder, dynamic power saving is 44.2% and leakage power saving is 4.74% at the optimal datapath width of 19bits; for MPEG-2 AAC audio decoder, the dynamic power saving is 14.5% and leakage power saving is 18.1% at the optimal datapath width of 24bits and MPEG2 video decoder, the dynamic power saving is 18.3% and leakage power is 19.1% at the optimal datapath width of 28bits. For different application, the number of variables is different and the effective size of variables is also different, therefore the optimal datapath width of minimal power is different. Note that this is under the assumption *ActTime* : *InactTime* = 1 : 1. *ActiTime* is the application execution time, which is called active time and *InactTime* is the idle time, which is called inactive time.

6.5 Commercial Products

In [Mutoh1996], they presented a power management processor, which uses MTCMOS technology.

Toshiba used the mixed MTCMOS and Dual V_T method to reduce the leakage power in a DSP core for W-CDMA cell phones. Cell phones spend a significant amount of time in the standby mode. Toshiba also presented a low power single-chip MPEG4 video-phone LSI. The VTCMOS technology is employed to reduce a standby leakage current, which is only 17% of the conventional CMOS design [ISSCC A 60MHz 240mW MPEG-4 videophone LSI with 16Mbit embedded DRAM].

6.6 Conclusions

This section describes leakage power reduction methodologies. There are four basic strategies: using high-V_T on non-critical paths, shifting low leakage mode, cooling high temperature parts, and reducing the number of transistors.

7. POWER REDUCTION TECHNIQUES USING APPLICATION SPECIFIC HARDWARE

The ultimate way for energy reduction is creating application-specific integrated circuits (ASICs) that implement their algorithms directly in dedicated, fixed-function logic. The most energy-efficient type of processor core is the “application-specific instruction processor” (ASIP). These processors are custom designed for the application at hand. Today, however, a few companies offer automated tools that generate ASIPs based on parameters supplied by the system designer. ASIC designers can also achieve good energy efficiency by starting with a processor core and then customizing the core to the needs of their application. The processor cores offered by ARC and Tensilica are specifically designed for customization by the system designer. Both companies' offerings allow the system designer to add custom instructions that can produce massive energy efficiency gains.

7.1 Energy-Flexibility Tradeoff

Power consumption heavily depends on an implementation style and its flexibility [Rabaey00]. In Figure 27, the tradeoff between energy consumption and flexibility for different architectures is shown. As one can see, the dedicated hardware (ASICs) is 4 orders of magnitude more power efficient than embedded processors. Therefore, if there is no need for flexibility, the ASIC implementation is preferred. In practice, however, many systems require flexibility of the system in order to support not only existing applications but also upcoming ones.

We can broadly categorize system architectures which concurrently satisfy high flexibility and low energy consumption as follows,

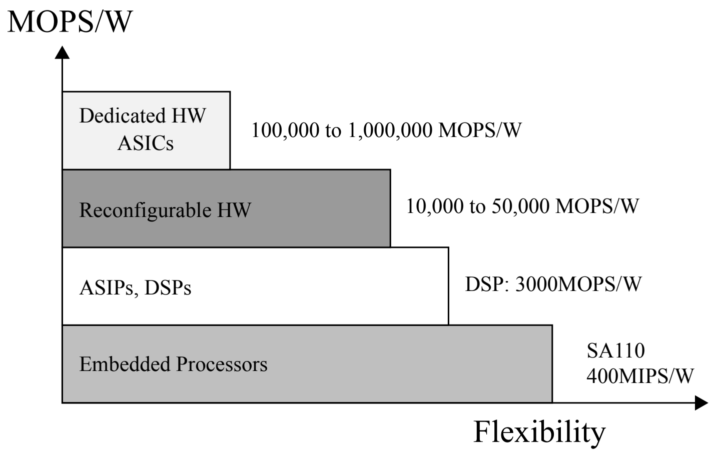


Figure 27. Energy-Flexibility Tradeoff

1. A hybrid architecture which consists of embedded processor or DSP and dedicated hardware, and
2. A configurable processor.

7.2 Hybrid Architecture

A hybrid-architecture consists of a microprocessor core, a set of standard cores, and a set of application specific cores as shown in Figure 28. The design goal using the hybrid-architecture is to partition a given application into the microprocessor core and the application specific cores in order to minimize the total energy consumption.

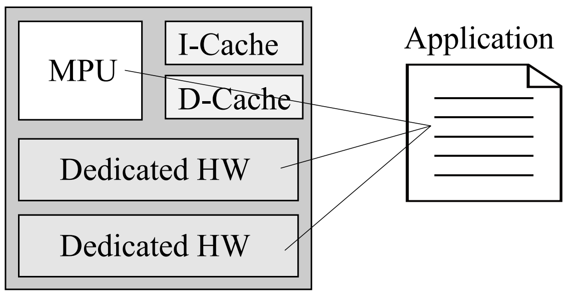


Figure 28. An Example of a Hybrid-Architecture

Hardware/software partitioning is the process of dividing an application into software running on a microprocessor and dedicated hardware. This approach is a well-established design methodology with the goal to increase the performance and to decrease the energy consumption of a system as described.

Dave et al. proposed a hardware/software co-design technique, called COSYN, which targets embedded systems consisting of general-purpose processors, ASICs and FPGAs [Dave97]. Functions of COSYN include allocation, scheduling, performance estimation, and power optimization. COSYN finds hardware/software partitioning based on the performance and power estimation of a processing element.

Henkel proposed a hardware/software partitioning technique for low-power core-based systems [Henkel99]. The technique considers the power consumption of a whole embedded system consisting of a microprocessor core, application specific cores, cache cores and a memory core. The technique is based on a fine-grained (instruction/operation-level) analysis of energy consumption. The experimental results demonstrated high reductions of power consumption between 35% and 94% at the cost of a relatively small additional hardware overhead.

7.3 Configurable Processor

A configurable processor core is a fully functional processor design that can be customized or expanded to meet the performance and/or energy efficiency needs of applications [Wei05]. There are four general ways a processor can be configured:

- By selecting from standard configuration options, such as bus widths, interfaces, memories, floating-point units, etc.
- By adding custom instructions that describe new registers, register files and custom data types, such as 56-bit data for security processing or 256-bit data types for packet processing.
- By adding custom, high-performance interfaces that exceed the bandwidth abilities of the more common shared-bus architectures of conventional RISC and DSP cores.

Configurable processors are typically delivered as synthesizable RTL code, and can be easily mapped onto an FPGA or SoC design. Some configurable processors are provided with automatically tailored software-development tools (the compiler, assembler, debugger, linker, and profiler), EDA synthesis scripts, and verification test benches that reflect the designer-defined architectural extensions so that no additional effort is required to ready the configured core for SoC development.

The ability to add custom instructions of any width allows an SoC designer to use a configurable processor core to implement datapath operations that closely match the abilities of a manually designed RTL block. Since the configurable processor does not have a feature for dynamically reconfiguring the structure of the processor, it is more energy efficient than a reconfigurable processor. In the configurable processor core, the datapaths are implemented using the base processor's integer pipeline, plus additional execution units, registers, and other functions added by the chip architect or SoC designer for a target application.

Energy efficiency of the configurable processor typically comes from the following three features [Wei05],

1. Configuration of the instruction set permits a much closer fit of the processor to the target application,
2. Configuring the processor removes unneeded hardware features like larger cache memories than needed, unused register files and extra bits of datapath [Inoue00], and
3. Automatic processor generation tools enable logic optimization, signal switching activity reduction, and seamless mapping into low-voltage circuits.

A lot of configurable processors and their optimization methodologies are proposed. However, only a few of them focus on methodologies for lowering energy consumption.

In [Inoue00], Inoue et al. proposed a flexible SoC architecture and its optimization framework, called FlexSys, which allows system designers to customize datapath width and memory size for a target application. A key of the FlexSys technology is that it allows designers to customize the core processor for the target application by replacing a few photomasks used for via layers only, which results in a low-cost customization of the processor for a target application. The experiments using DSPstone benchmark programs demonstrated that the energy consumption can be reduced by 54% compared to the normal RISC processor-based system which has a CPU core with 32-bit datapath and the fixed number of memory words.

7.4 Conclusions

In this section we introduced a concept of energy-flexibility tradeoff. We showed that system designers can drastically reduce energy consumption by trading flexibility for energy consumption. However, in practice, it is very important to preserve system flexibility in case of future upgrade or modification in a target application. Therefore, we have to find the best compromising point between high flexibility and low energy consumption. We can broadly categorize system-level methodologies which satisfy high flexibility with low energy consumption as follows,

1. hardware/software partitioning for a hybrid architecture which consists of a microprocessor core and dedicated hardware and
2. exploiting customizability of configurable processors.

These strategies allow system designers to explore SoC architectures considering tradeoff between flexibility and energy consumption. As a result, system designers can find the best tradeoff point which compromises between high flexibility and low energy consumption.

8. SUMMARY

This chapter addressed several key methodologies for reducing power and/or energy consumption of SoCs which consist of hardware and software running on it. Each of those methodologies takes design tradeoffs into consideration. In Section 3, we introduced an energy-delay tradeoff and an energy-reliability tradeoff in SoC design. Section 4 discussed on a tradeoff between energy consumption and quality of services (QoS). The QoS, in this chapter includes precision (or computational quality) and latency (or response time). In section 5, a tradeoff between computational energy and communication energy is considered. Section 6 summarized several leakage reduction techniques considering the energy-delay tradeoff and the energy-QoS tradeoff. In Section 7, we introduced an energy-flexibility tradeoff. The key point of the energy reduction techniques is to take the tradeoffs into consideration according to a design objective and design constraints.

The problem of how to model and evaluate complicated SoCs in terms of energy, performance, QoS, reliability and flexibility becomes more attractive to tackle. As the supply voltage and threshold voltage of chips is lowered down along with the transistor scaling, sensitivity to temperature variation, process variation, sources of soft error and noise sources is increased. This results in model uncertainty and makes evaluation of SoC difficult. Increasing size, complexity, and functionality integrated on SoC causes this problem to become more difficult. In the near future, modeling and evaluation of SoC dynamically and/or statically taking the model uncertainty into account is one of the most important themes for low-energy SoC design.

REFERENCES

[Black69] J. R. Black, "Electromigration Failure Modes in Aluminum Metallization for Semiconductor Devices, " in Proc. of IEEE, vol. 57, no. 9, pp. 1587-1594, Sep. 1969.

[Weste93] N. Weste and K. Eshraghian, “Principles of CMOS VLSI design”, Addison-Wesley, 1993.

[Chatterjee96] A. Chatterjee, M. Nandakumar, and I. Chen, “An Investigation of the Impact of Technology Scaling on Power Wasted as Short-Circuit Current in Low Voltage Static CMOS Circuits,” in Proc. ISLPED, pp. 145-150, Aug. 1996.

[Usami95] K. Usami, and M. Horowitz, “Clustered Voltage Scaling Techniue for Low-Power Design”, in Proc. of Int’l Symposium on Low Power Design, pp. 3-8, April, 1995.

[Johnson97] M. C. Johnson and K. Roy, “Datapath Scheduling with Multiple Supply Voltages and Level Converters,” ACM TODAES, vol.2, no.3, pp. 227-248, July, 1997.

[Chandrakasan95] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. W. Brodersen, “Optimizing Power Using Transformations,” IEEE Trans. on CAD, vol.14, no.1, pp. 12-31, Jan., 1995.

[Raghunathan94] A. Raghunathan and H. K. Jha, “Behavioral Synthesis for Low Power”, in Proc. of Int’l Conference on Computer Design, pp. 318-322, Oct., 1994.

[Raghunathan95] A. Raghunathan and H. K. Jha, “An Iterative Improvement Algorithm for Low Power Data Path Synthesis”, in Proc. of Int’l Conference on Computer Aided Design, pp. 597-602, Nov., 1995.

[Goodby94] L. Goodby, A. Orailoglu, and P. M. Chau, “Microarchitectural Synthesis of Performance-Constrained Low-Power VLSI Designs”, In Proc. of Int’l Conference on Computer Design, pp. 323-326, Oct., 1994.

[Kumar95] N. Kumar, S. Katkoori, L. Rader, and R. Vemuri, “Profile-Driven Behavioral Synthesis for Low-Power VLSI Systems”, IEEE Design & Test, vol.12, no.3, pp. 70-84, Fall, 1995.

[Martin95] R. S. Martin, and J. P. Knight, “Power-Profiler: Optimizing ASICs Power Consumption at the Behavioral Level”, In Proc. of Design Automation Conference, pp. 42-47, June, 1995.

[Raje95] S. Raje, and M. Sarrafzadeh, “Variable Voltage Scheduling”, in Proc. of Int’l Symposium on Low Power Design, pp. 9-14, April, 1995.

[Lin97] Y. R. Lin, C. T. Hwang and A. C.-H. Wu, “Scheduling Techniques for Variable Voltage Low Power Designs”, ACM TODAES, vol.2, no.2, pp. 81-97, April, 1997.

[Chang96] J. Chang and M. Pedram, “Energy Minimization Using Multiple Supply Voltages”, in Proc. of Int’l Symposium on Low Power Electronics and Design, pp. 157-162, Aug., 1996.

[Ishihara98] T. Ishihara, and H. Yasuura, “Voltage Scheduling Problem for Dynamically Variable Voltage Processors”, in Proc. of Int’l Symposium on Low Power Electronics and Design, pp. 197-202, Aug. 1998.

[Weiser94] M. Weiser, B. Welch, A. Demers and S. Shenker, “Scheduling for Reduced CPU Energy”, in Proc. of Symposium on Operating Systems Design and Imprementation, pp. 13-23, Nov., 1994.

[Yao95] F. Yao, A. Demers and S. Shenker, “A Scheduling Model for Reduced CPU Energy”, in Proc. of Symposium on Faundations of Cumputer Science, pp. 374-382, Oct., 1995.

[Shin99] Y. Shin and K. Choi, “Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems”, in Proc. of Design Automation Conference, pp. 134-139, June, 1999.

[Shin00] Y. Shin, K. Choi and T. Sakurai, “Power Optimization of Real-Time Embedded Systems on Variable Speed Processors”, in Proc. of Int’l Conference on Computer Aided Design, pp. 365-368, Nov., 2000.

[Okuma99] T. Okuma, T. Ishihara, and H. Yasuura, “Real-Time Task Scheduling for a Variable Voltage Processor”, in Proc. of Int’l Symposium on System Synthesis, pp. 24-29, Nov., 1999.

[Austin04] T. Austin, D. Blaauw, T. Mudge and K. Flautner, “Making Typical Silicon Matter with Razor”, IEEE Computer Magazien, pp. 57-65, March 2004.

[Bertozzi02] D. Bertozzi, L. Benini and G. De Micheli, “Low-Power Error-Resilient Encoding for On-Chip Data Busses”, in Proc of Dasign Automation and Test in Europe Conference, pp. 102-109, March, 2002.

[Worm02] F. Worm, P. Lenne, P. Thiran and G. De Micheli, “An adaptive low-power transmission scheme for on-chip networks”, in Proc. of Int’l symposium on system synthesis, pp. 92-100, Oct. 2002

[Cao2003] Y. Cao, and H. Yasuura, “Quality-Driven Design by Bitwidth Optimization for Video Applications,” in Proc. Asia and South Pacific Design Automation Conference, pp. , 2003.

[Alalusi2000] S. Alalusi, and B. Victor, “Variable Word Width Computation for Low Power,” CS 252 Computer Architecture, 2000.

[Stephenson2000] M. Stephenson, J. Babb, and S. Amarasinghe, “Bitwidth Analysis with Application to Silicon Compilation,” in Proc. ACM SIGPLAN 2000 Conference on Programming language design and implementation, pp. 198-120, 2000.

[Canal2000] R. Canal, A. Gonzalez, and J. E. Smith, “Very Low Power Pipelines using Significance Compression,” in Proc. of International Symposium on Microarchitecture, pp. 181-190, 2000.

[Brooks2000] D. Brooks, and M. Martonosi, “Value-Based Clock Gating and Operation Packing: Dynamic Strategies for Improving Processor Power and Performance,” in ACM Transactions on Computer Systems, vol. 18, no. 2, pp. 89-126, May 2000.

[Bhunia2003] H. Li, S. Bhunia, Y. Chen, T. N. Vijaykumar, and K. Roy, “Deterministic Clock Gating for Microprocessor Power Reduction,” in Proc. of International Symposium on High-Performance Computer Architecture, pp. 113, 2003.

[Bellas1999] N. Bellas, I. Haji, and C. Polychronopoulos, “Using Dynamic Cache Management Techniques to Reduce Energy in a High-Performance Processor,” in Proc. of International Symposium on Low Power Electronics and Design, pp. 64-69, 1999.

[Benini1998] L. Benini, A. Bogliolo, S. Cavallucci, and B. Ricco, “Monitoring Systems Activity or OS-Directed Dynamic Power Management,” in Proc. of International Symposium on Low Power Electronics and Design, pp. 185-190, 1998.

[Wong2004] J. L. Wong, G. Qu, and M. Potkonjak, “Power Minimization in QoS Sensitive Systems,” in IEEE Transactions on Very Large Scale Integration Systems, vol. 12, no. 6, pp. 553-561, 2004.

[Qiu2001] Q. Qiu, Q. Wu, and M. Pedram, “Dynamic Power Management in a Mobile Multimedia System with Guaranteed Quality-of-Service,” in Proc of Design Automation Conference, pp. 834-839, 2001.

[Yardi2005] S. M. Yardi, M. S. Hsiao, T. L. Martin, and D. S. Ha, “Quality-Driven Proactive Computation Elimination for Power-Aware Multimedia Processing,” in Proc of DATE, pp. 340-345 , 2005.

[Pokam2004] G. Pokam, O. Rochecouste, A. Seznec, and F. Bodin, “Speculative Software Management of Datapath-width for Energy Optimization,” in Proc. of LCTES, pp. 78-87, 2004.

[Sinha2002] A. Sinha, A. Wang, and A. Chandrakasan,“Energy Scalable System Design,” in IEEE Transactions on Very Large Scale Integration Systems, vol. 10, no. 2, 2002.

[Bellosa1999] F. Bellosa, “OS-Directed Throttling of Processor Activity for Dynamic Power Management,” Tech. Report, 1999.

[Muroyama2003] M. Muroyama, A. Hyodo, T. Okuma, and H. Yasuura, “A Power Reduction Scheme for Data Buses by Dynamic Detection of Active Bits,” in Proc. of

Euromicro Symposium on Digital System Design - Architectures, Methods and Tools -, pp. 408-415, 2003.

[Okuma2002] T. Okuma, Y. Cao, M. Muroyama, and H. Yasuura, “Reducing Access Energy of On-Chip Data Memory Considering Active Data Bitwidth,” in Proc. of International Symposium on Low Power Electronics and Design, pp. 88-91, 2002.

[Xanthopoulos2000] T. Xanthopoulos, and A. P. Chandrakasan, “A Low-Power DCT Core Using Adaptive Bitwidth and Arithmetic Activity Exploiting Signal Correlations and Quantization,” in IEEE Jounal of Solid-State Circuits, vol. 5, no. 5, 2000.

[Liu94] D. Liu and C. Svensson, “Power Consumption Estimation in CMOS VLSI Chips,” IEEE Journal of Solid-State Circuits, vol.29, no.6, pp. 663-670, June 1994.

[Ko98] U. Ko, P. T. Balsara, and A. K. Nanda, “Energy Optimization of Multilevel Cache Architectures for RISC and CISC Processors,” IEEE Trans. on VLSI Systems, vol.6, no.2, pp. 299-308, June 1998.

[Su95] C. L. Su and A. M. Despain, “Cache Design Trade-offs for Power and Performance Optimization: A Case Study”, in Proc. of ISLPED, pp. 63-68, August 1995.

[Hicks97] P. Hicks, M. Walnock, and R. M. Owens, “Analysis of Power Consumption in Memory Hierarchies”, in Proc. of ISLPED, pp. 239-242, August 1997.

[Li98] Y. Li, and J. Henkel, “A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems”, in Proc. of DAC, pp. 188-193, June, 1998.

[Shine99] W. T. Shine, and C. Chacrabarti, “Memory Exploration for Low Power, Embedded Systems”, in Proc. of DAC, pp. 140-145, June, 1999.

[Malik00] A. Malik, B. Moyer and D. Cermak, “A Low Power Unified Cache Architecture Providing Power and Performance Flexibility”, in Proc. of ISLPED, pp. 241-243, July 2000.

[Ishihara05] T. Ishihara and F. Fallah, “A Non-Uniform Cache Architecture for Low Power System Design”, in Proc. of ISLPED, pp. 363-368, Aug., 2005.

[McFarling89] S. McFarling, “Program Optimization for Instruction Caches”, In Proc. of Int’l Conference on Architecture Support for Programming Languages and Operating Systems, pp. 183-191, April 1989.

[Hwu89] W. W. Hwu and P. P. Chang, “Achieving High Instruction Cache Performance with an Optimizing Compiler”, in Proc. of ISCA, pp. 242-251, May 1989.

[Tomiyama96] H. Tomiyama and H. Yasuura, “Optimal Code Placement of Embedded Software for Instruction Caches”, in Proc. of European Design and Test Conference, pp. 96-101, March, 1996.

[Panda96] P. Panda, N. Dutt, and A. Nicolau, “Memory Organization for Improved Data Cache Performance in Embedded Processors”, in Proc. of the 9th Int’l Symposium on System Synthesis, pp. 90-95, November 1996.

[Hashemi97] A. H. Hashemi, D. R. Kaeli, and B. Calder, “Efficient Procedure Mapping Using Cache Line Coloring”, in Proc. of Programming Language Design and Implementation, pp. 171-182, June, 1997.

[Ghosh99] S. Ghosh, M. Martonosi, and S. Malik, “Cache Miss Equations: A Compiler Framework for Analyzing and Tuning Memory Behavior”, ACM Trans. on Programming Languages and Systems, vol.21, no.4, pp. 703-746, July, 1999.

[Kulkarni01] C. Kulkarni, C. Ghez, M. Miranda, F. Catthoor, H. De Man, “Cache Conscious Data Layout Organization for Conflict Miss Reduction in Embedded Multimedia Applications,” in Proc. of DATE 2001, pp. 686-691, March 2001.

[Bankar02] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, “Scratchpad Memory : A Design Alternative for Cache On-Chip Memory in Embedded Systems”, in Proc. of CODES, pp. 73-78, May, 2002.

[Stan95] M. R. Stan and W. P. Burleson, "Bus-Invert Coding for Low-Power I/O," IEEE Trans. on VLSI Systems, vol. 3, pp. 49-58, March, 1995.

[Su94] C. L. Su, C. Y. Tsui, and A. M. Despain, "Low Power Architecture Design and Compilation Technique for High-Performance Processors," in Proc. IEEE COMPCON, pp. 209-214, Feb., 1994.

[Benini97] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano, "Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems," in Proc. of Great Lakes Symposium on VLSI, pp. 77-82, March, 1997.

[Benini97-2] L. Benini, G. De Micheli, E. Macii, M. Poncino, and S. Quer, "System-Level Power Optimization of Special Purpose Applications: The Beach Solution," in Proc. of Int'l Symposium on Low Power Electronics and Design, pp. 24-29, August, 1997.

[Shin98] Y. Shin, S. Chae, and K. Choi, "Partial Bus-Invert Coding for Power Optimization of System Level Bus," in Proc. of Int'l Symposium on Low Power Electronics and Design, pp. 127-129, August, 1998.

[Benini00] L. Benini, A. Macii, E. Macii, and M. Poncino, "Synthesis of Application Specific Memories for Power Optimization in Embedded Systems", in Proc. of Design Automation Conference, pp. 300-303, June, 2000.

[Ishihara00] T. Ishihara, and H. Yasuura, "A Power Reduction Technique with Object Code Merging for Application Specific Embedded Processors", in Proc. of Design Automation and Test in Europe Conference, pp. 617-623, March, 2000.

[Segars95] S. Segars, K. Clarke, L. Goudge, "Embedded Control Problems, Thumb and the ARM7TDMI," IEEE Micro, vol.15, no.5, pp. 22-30, Oct., 1995.

[Yoshida97] Y. Yoshida, B. Y. Song, H. Okuhata, T. Onoye, and I. Shirakawa, "An Object Code Compression Approach to Embedded Processors," in Proc. of Int'l Symposium on Low Power Electronics and Design, pp. 285-288, August, 1997.

[Benini99] L. Benini, A. Macii, E. Macii, and M. Poncino, "Selective Instruction Compression for Memory Energy Reduction in Embedded Systems", in Proc. of Int'l Symposium on Low Power Electronics and Design, pp. 206-211, August, 1997.

[Powell2000] M. D. Powell, S. Yang, B. Falsafi, K. Roy, T. N. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," in Proc. of International Symposium on Low Power Electronics and Design, pp. 90-95, 2000.

[Yan2005] L. Yan, J. Luo, and N. K. Jha, "Joint Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-Time Embedded Systems," IEEE Trans. on CAD, vol.24, no.7, pp. 1030-1041, July 2005.

[Tsai2004] Y.-F. Tsai, D. E. Duarte, N. Vijaykrishnan, and M. J. Irwin, "Characterization and Modeling of Run-Time Techniques for Leakage Power Reduction," in IEEE Transactions on Very Large Scale Integration Systems, vol. 12, no. 11, 2004.

[Cao2002] Y. Cao, and H. Yasuura, "Leakage Power Reduction Using Bitwidth Optimization," in Proc. of the 6th World Multiconference on Systemics, Cybernetics and Informatics., 2002.

[Li2005] P. Li, Y. Deng, and L. T. Pileggi, "Temperature-Dependent Optimization of Cache Leakage Power Dissipation," in Proc. of ICCD, 2005.

[Khouri2002] "Leakage Aware Synthesis," in Proc. of TVLSI 2002.

[Ynag2001] P. Yang, C. Wung,..."Energy-Aware Runtime Scheduling for Embedded Multiprocessor SoCs," in IEEE Design and Test of Computers, 2001.

[Ishihara2002] T. Ishihara, and K. Asada, "An architectural level energy reduction technique for deep-submicron cache memories," in Proc. of Asia and South Pacific Design Automation Conference, 2002.

[Kaxiras2000] S. Kaxiras, Z. Hu, G. Narlikar, and R. McLellan, "Cache-line decay: a mechanism to reduce cache leakage power," in Proc. of Workshop on Power Aware Computer Systems, 2000.

[Manne1998] S. Manne, A. Klauser, and D. Grunwald, "Pipeline gating: speculation control for energy reduction," in Proc. of International Symposium on Computer Architecture, 1998.

[Sato2004] H. Sato, and T. Sato, "A Static and Dynamic Energy Reduction Technique for I-Cache and BTB in Embedded Processors," in Proc. of Asia South Design Automation Conference, 2004.

[Schmidt2002] R. Schmidt, and B. Notohardjono, "High-end Server Low-Temperature Cooling," in IBM Journal of Research and Development, pp. 739-751, 2002.

[Krane1988] R. Krane, J. Parsons, and A. Bar-Cohen, "Design of a candidate thermal control system for a cryogenically cooled computer," in IEEE Transactions on Components, Hybrids, and Manufacturing Technology, vol. 11, no. 4, pp. 545-556, 1988.

[Calhoun2003] B. H. Calhoun, F. A. Honore, and A. Chandrakasan, "Design methodology for fine-grained leakage control in MTCMOS," in Proc. of International Symposium on Low Power Electronics and Design, pp. 104-109, 2003.

[Liu2004] M. Liu, W.-S. Wang, and M. Orshansky, "Leakage Power Reduction by Dual-Vth Designs under Probabilistic Analysis of Vth Variation," in Proc. of International Symposium on Low Power Electronics and Design, pp. 2-7, 2004

[Wei1998] L. Wei, Z. Chen, M. Johnson, K. Roy, and V. De, "Design and Optimization of Low Voltage High Performance Dual Threshold CMOS Circuits," in Proc. of Design Automation Conference, pp. 489-494.

[Ho2004] Y.-T Ho, and T.-T. Hwang, "Low Power Design using Dual Threshold Voltage," in Proc. of Asia and South Pacific Design Automation Conference, pp. 205-208, 2004.

[Seta1995] K. Seta, H. Hara, T. Kuroda, M. Kakumu, and T. Sakurai, "50% Active-Power Saving without Speed Degradation using Standby Power Reduction (SPR) circuit," in Proc. of ISSCC, pp. 318-319, 1995.

[Kobayashi1994] Kobayashi, and T. Sakurai, "Self-Adjusting Threshold Voltage Scheme (SATS) for Low-Voltage High-Speed Operation," in Proc. of CICC, pp. 271-274, 1994.

[Claasen] T. Claasen,... in Proc. of ISSCC99.

[Budiu2002] M. Budiu, "Application-Specific Hardware," in Proc. International Conference on Field Programmable Logic and Applications, pp. 853-863, 2002.

[Ranpara1999] S. Ranpara et al., "A Low-Power Viterbi Decoder Design for Wireless Communications Applications," in Proc. ASIC, 1999.

[Gilbert2001] F. Gilbert et al. "Low Power Implementation of a Turbo-Decoder on Programmable Architectures" in Proc. ASP-DAC, 2001.

[Usami] K. Usami et al. "Design Methodology of Ultra Low-power MPEG4 Codec Core Exploiting Voltage Scaling Techniques"

[Lee2003] R. B. Lee, "Challenges in the Design of Security-Aware Processors," in Proc. Application-Specific Systems, Architectures, and Processors, pp. , 2003.

[Udayakumaran2002] S. Udayakumaran, B. Narahari, R. Simha, "Application-Specific Memory Partitioning for Low Power Consumption," COLP 2002.

[Rabaey00] J. M. Rabaey, "Low Power Silicon Architecture for Wireless Communications," in Proc. of Asia South Pacific Design Automation Conference, pp. 377-380, January, 2000.

[Wei05] J. Wei, and C. Rowen, "Implementing Low-Power Configurable Processors – Practical Options and Tradeoffs," in Proc. of Design Automation Conference, pp. 706-711, June, 2005.

[Dave97] B. P. Dave, G. Lakshminarayana, and N. K. Jha, “COSYN: Hardware-Software Co-Synthesis of Embedded Systems,” in Proc. of Design Automation Conference, pp. 703-708, June, 1997.

[Henkel99] J. Henkel, “A Low Power Hardware/Software Partitioning Approach for Core-based Embedded Systems,” in Proc. of Design Automation Conference, pp. 122-127, June, 1999.

[Inoue00] A. Inoue, T. Ishihara, and H. Yasuura, “Flexible System LSI for Embedded Systems and Its Optimization Techniques”, in H. Yasuura, editors, Journal of Design Automation for Embedded Systems, Vol.5, No.2, pp. 179-205, Kluwer Academic Publishers, Jun. 2000.

Chapter 7

SoC PROTOTYPING AND VERIFICATION

Moo-Kyoung Chung[†], Young-Il Kim[‡], Jae-Gon Lee[†], Wooseung Yang[‡], Ando Ki[‡], and Chong-Min Kyung[†]
[†] Korea Advanced Institute of Science and Technology; [‡] Dynalith Systems Co., Ltd.

Abstract: Verification of System-On-a-Chip (SoC) poses us a serious challenge as it involves not only high chip complexity but also hardware/software co-verification along with short design time-to-market. Traditional IC design verification technologies based on simulation, emulation, and prototyping often fall short of meeting this challenge of SoC verification. This chapter starts with an introduction of SoC design verification flow. To reduce the time-to-market it is crucial to provide the system-level model for each hardware block, software component and communication channel in the very early stage of the SoC design process. It can be best addressed by performing the so-called 'soft prototyping.' System-level modeling using SystemC is explained as it is expected to be widely employed as a reference model. Software part of the SoC is run on Instruction Set Simulation (ISS), which is interfaced to hardware models described in either software (like HDL or SystemC) or physical hardware. We explained the hybrid SoC design verification technique which incorporates both simulation and prototyping in a single verification environment to maximally exploit the merits of both approaches. Simulation acceleration and emulation are explained followed by the introduction of HW/SW co-simulation and FPGA-based co-emulation techniques. These techniques based on initial system-level modeling of high-level abstract behavior followed by gradual refinement and verification by comparing with the reference model, enables fast and error-free SoC design closure

Keywords: SoC, Prototype, Verification, Soft Prototype, Hard Prototype, Co-Simulation, Instruction Set Simulator (ISS)

1. INTRODUCTION

There are clear trends in SoC design; increasing chip size, increasing complexity of functionality, embedding processing cores, supporting multi-media features, decreasing time-to-market, and decreasing power consumption. Due to increasing size and complexity, pure software-based simulation easily fails to deliver sufficient performance. Thus, speed-up techniques such as raising abstract level or adopting hardware accelerator are necessary. Due to embedding processing cores, hardware and software co-verification is inevitable. Due to supporting multi-media features, more realistic test-bench is required, which sometimes includes real hardware. Due to decreasing time-to-market, validation of functions and features and verification of chosen architecture must be done at early design stage. Due to decreasing power consumption, low power design and optimum hardware-software partitioning need to be considered as well.

Figure 1 depicts an SoC design flow and its verification environment. The first step is idea validation where idea has to be validated in terms of technology viability, resource availability, and market requirements. The second step is system-level design where all functionalities are considered without notion of hardware and software and, then, possible architectures are exploited in order to choose optimum or reasonable one. After fixing architecture, hardware and software developments are carried out in parallel. Other steps are fairly similar to the conventional design flow.

In order to carry out system-level design and hardware-software co-development, SoC model is required, where the SoC model should be flexible enough to easily change its configuration, provide full visibility to monitor its behavior and provide performance statistics. The SoC model in these steps is called 'soft prototype', in which most components are modeled using software including C/C++, SystemC, HDL and so on. As a design makes its progress, abstraction level of blocks is lowered. This causes the software prototype to run slow as the lower-level model entails more details. In order to cope with this performance degradation problem, hardware-assisted

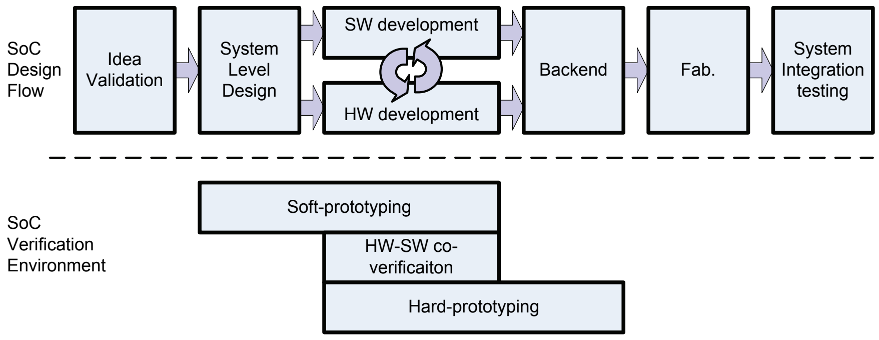


Figure 1. SoC design flow and verification environment

approaches are widely used, e.g., hardware acceleration, hardware emulation, and even FPGA prototyping. In this context, the term 'hard prototype' is used for solutions utilizing hardware. In the heart of hardware-software co-verification, hardware-software co-simulation is one of the most important techniques since it provides an environment of running software on top of model of SoC hardware sub-system.

Although the top-down approach as shown in Figure 1 starting from system specification down to gate-level and executable code level through hardware-software co-development is very logical, meet-in-the-middle approach is more widely used in SoC design. This is so called platform-based design (Chang et al., 1999), where platform is a pre-designed architecture that designers can use to build SoC for a given range of applications. Therefore, a large portion of SoC design is not designed from scratch, but rather often derived from pre-designed ones. Building soft and hard prototypes can be built from the platform.

This chapter covers the following topics in the context of SoC prototyping and verification. Firstly, soft prototyping is explained. Secondly, hardware-software co-verification is addressed. Thirdly, hard prototyping is presented.

2. SOFT PROTOTYPING

The first step in designing SoC is to build soft-model of the target system in the early design stage. As can be noticed from the notation, it is mainly built as a form of a software-based simulation. This method is fast and efficient to build and easy to manipulate, which makes it best for early-stage prototyping where lots of design turn-around is mandatory. In this chapter, we use the term 'soft prototyping' for this stage. In the past, there was no golden rule for building soft prototyping. Each designer or design team built their soft prototypes for his/her particular purpose. Usually they are based on programming languages, typically C-based languages. But as there were no standards for the use of the languages, soft prototypes from different teams couldn't run together. Recently, the introduction of SystemC standardized the soft prototyping stage, which made it possible to encapsulate highly abstracted IP models just like we handle RTL IP cores. This chapter starts from introduction to SoC design flow and handles major issues of soft prototyping.

2.1 SoC Design Flow

The design step of SoC is composed of several steps with different abstraction levels. The design starts with a highly abstracted description of the target system and is refined until the hardware part is ready for synthesis and the software part is ready for compilation. Figure 2 shows the conventional design steps. SoC design starts with a specification that roughly describes the operations and performance requirements of the target SoC. The specification is usually described in natural languages with block diagrams, tables and mathematical equations. The process of obtaining the specification is far from automatic procedures commonly found in the following design steps. Rather, it is a manual process with discussions and speculations among engineers.

When the specification is fixed, the algorithm of the target system is verified. This is called algorithm-level reference model verification. This model confirms the algorithm of the target SoC. It is described with programming languages, which can concentrate on the flow of data not worrying about its implementation. Among many programming languages C and C++ are the most popular as many engineers are familiar with the two

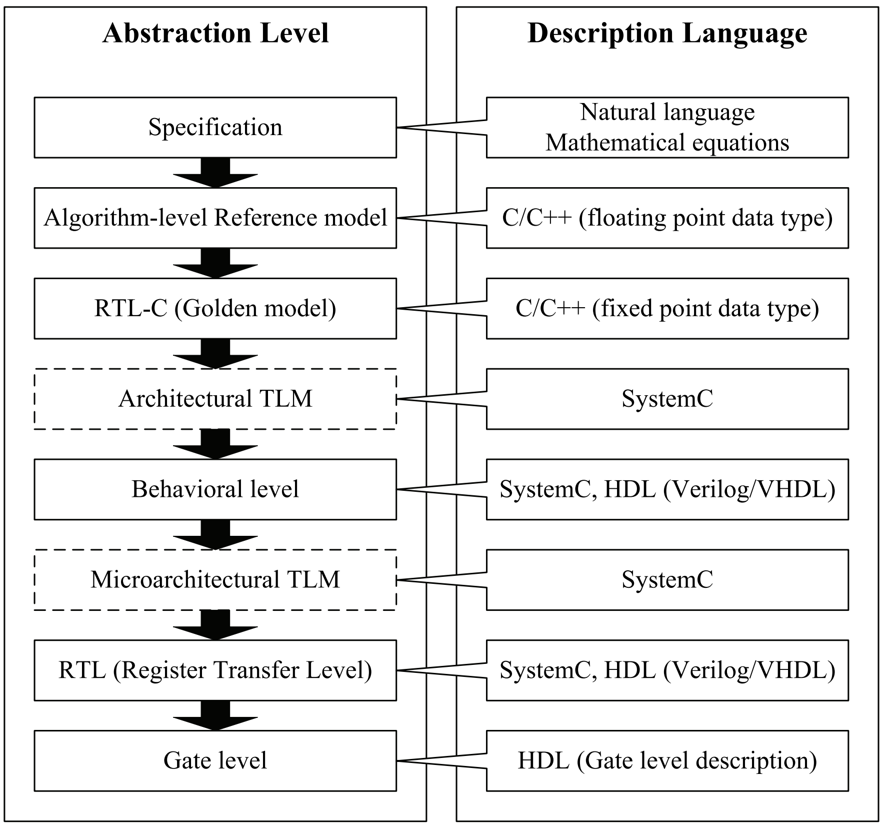

Abstraction Level
Description Language
Specification
Natural language
Mathematical equations
Algorithm-level Reference model
C/C++ (floating point data type)
RTL-C (Golden model)
C/C++ (fixed point data type)
Architectural TLM
SystemC
Behavioral level
SystemC, HDL (Verilog/VHDL)
Microarchitectural TLM
SystemC
RTL (Register Transfer Level)
SystemC, HDL (Verilog/VHDL)
Gate level
HDL (Gate level description)

Figure 2. SoC Design steps and description languages

languages, while MATLAB is also frequently used. In algorithm-level reference model, floating point data types that are used as the flow of data, not its implementation, is of major concern.

The next step is to validate the implementation of the algorithm. It is called RTL-C, or more often as “Golden model.” RTL-C roughly models the structure of the target system with programming languages. Unlike algorithm-level reference model, fixed point data types are used in RTL-C. Simulation time can be introduced here even though cycle-accurate behavior is not yet modeled. Usually, RTL-C model is used as a reference model for the following design steps, especially RTL design, which requires lots of design time and efforts.

In the conventional design steps, the next step is behavioral level description. Behavioral level description models the behavior, or interfaces, of each building block. The structure of each block is not modeled in behavioral level description. The simulation time is modeled in a cycle-accurate manner in this level. High-level synthesis tools provide automatic translation from behavioral level description to gate level netlist. But, usually behavioral level description is further refined to RTL for better results. This is because high-level synthesis tools cannot as efficiently utilize the time and space as RTL model. RTL model describes all the internal and external structure of each block in a full cycle accurate manner. It is ready to be synthesized to gate level netlist.

With conventional design steps we can get cycle-accurate model only after behavior-level description is obtained. As behavioral level descriptions are not used very often, it can be as late as until RTL model is available. In other words, we cannot have cycle-accurate SoC models in the early design steps. This can be a serious problem as we cannot predict the performance of the target system until significant design time is consumed. This is becoming a more serious problem as the portion of embedded software becomes larger. Even today, the embedded software accounts for more than half of the total expected functionality of the circuit and very often most of the modifications that occur during the design of a chip based on an existing platform are software updates as shown by Miller (2003). An obvious consequence of this is that the critical path for the development of such a circuit is the software, not the hardware. Enabling software development to start very early in the development cycle is, therefore, crucial to reduce the time-to-market.

There have been a number of attempts to solve this problem with conventional design steps. Firstly, algorithmic model was used for system level modeling. Even though algorithmic model runs extremely fast as it only captures the algorithm, algorithmic model does not have a notion of simulation time or concepts of hardware and software blocks. Secondly, C-based dialects with hardware modeling concepts emerged. This enabled cycle-accurate behavior in the early design steps. But, the cycle-accurate model ended up being only an order of magnitude faster than the equivalent RTL simulation, which is very similar to the speed of cycle based VHDL/Verilog. And much of the information captured in such a model was not available in the IP documentation but only in the designer's mind. The resulting model was in only slightly higher abstraction level than RTL. We need more abstracted model of the target system while still keeping cycle accuracy.

2.2 Transaction Level Modeling

Grotker (2003) defined Transaction-level modeling (TLM) as a style for modeling digital systems focusing on external functional behavior of each block and inter-block communications without excessive implementation details. The function stands for behavior or operation of each building block described in high abstraction level with variables, arithmetic operations and sub-routine calls. TLM does not directly handle signals and registers, although the variables may be correlated to signal values of RTL designs. Even if the high-level modeling of the operations of each block resembles that of untimed algorithm-level description, TLM is different from untimed algorithm-level description in that block boundaries are defined and communications between building blocks are explicitly declared with transactions and each building block is modeled as a separate module while communications are modeled with transactions between them.

Transaction is a high-level abstraction of transition activity of interface signals between two or more building blocks. When a block (A) needs to communicate with other block (B), the block A invokes transactions to be serviced by the block B. The transaction can be a read transaction or a write transaction. It can also be a read transaction and a write transaction at the same time.

Clouard et al. (2003) divided TLM into two categories, i.e., architectural TLM and micro-architectural TLM, according to the abstraction level of simulation time. In architectural TLM, simulation time is roughly modeled, which is, therefore, called pseudo cycle-accurate. Transactions in this method are highly-abstracted information about communications between blocks. This method is suitable for early-stage prototyping of SoC designs where cycle-accurate behavior is either undefined or unnecessary. In micro-architectural TLM, the simulation time is modeled in a fully cycle-accurate manner. In this method, transactions are communication events between blocks while they can be easily correlated to a single signal event in the RTL design. Single transaction of the architectural TLM can be decomposed into multiple micro-architectural transactions. With micro-architectural TLM, we can verify operations of SoC design in a 100% clock cycle-accurate manner in the early design stage when its RTL model is not yet available as noted by Caldari (2003).

2.2.1 SystemC

Conventional hardware description languages are not suitable for handling TLM. SystemC is a set of C++ class definitions and methodologies for using these classes. The primary goal of SystemC is to enable system-level modeling encompassing software algorithm, hardware architecture, and interfaces of SoC. SystemC was first released in 1999 by Open SystemC Initiative (OSCI) with version 0.9. Since then there were multiple releases. Now SystemC version 2.1 is available (OSCI, 2005). SystemC class library adds necessary components for system level modeling to standard C++ language. These include modeling of simulation time, concurrency, and reactive behavior. Each building block of the system is modeled with an object of a particular C++ class. This includes modules, channels, and ports. As lots of building blocks share common features, these C++ classes are constructed in a hierarchical way. At the same time, SystemC retains native features of C++ language. This means that system designer can easily merge SystemC environment with native C++ software environments. In addition, system designers who are already familiar with C++ can use SystemC without any additional training.

In SystemC, each building block is modeled as a module or a channel. Modules can call interfaces provided by channels. This is called interface method call (IMC), which corresponds to a single transaction. A module can be a channel, and a channel can be a module, too. Usually, a transaction service routine of a channel, also called an interface method, activates another transaction, which activates yet another transaction service routine, and so on. Like chain reactions, sequential activations of interface methods form a simulation.

With SystemC, system designers can quickly simulate their designs, validate and optimize them, explore various algorithms, and provide the hardware and software development teams with an executable specification of the system. The executable specification removes the gap between the literal specification and RTL implementation shown in Figure 2. In the conventional design flow, engineers read the literal specifications and manually convert them to implementation, which tends to introduce lots of mistakes and misunderstandings. This method also suffers from the fact that the system model is built on environments totally different from HDL environments. Finally, the conventional method needs multiple system verifications for each abstraction level. As test vectors that are created to validate the C model typically cannot be run against the HDL model, test vectors should be created for HDL model. SystemC design methodology offers many advantages by providing a unified environment for multiple abstraction levels. The specification itself is designed as an executable specification so that there is no misunderstanding of the specification from the beginning. As SystemC can cover abstraction levels from the specification to RTL models, high abstraction level models can be gradually refined during the design process. In addition, test vectors used in the early design stages can be reused in the final design stages as a single language is used.

2.2.2 Characteristics of TLM

TLM is useful for early stage system design. The system designer can verify cycle-accurate behavior of target system with transaction-level models long before RTL designs are available. As the implementation details are not handled in TLM, the simulation speed is much faster than that of RTL simulation. Transaction-level (TL) models typically run at least two orders of magnitude faster than RTL models as shown by Clouard (2002). Simulation speeds of several hundred kilocycles per second for a complete system simulation is readily achievable with TLM compared to several hundred cycles per second in RTL models as shown by Clouard et al. (2003). This means that by using TLM we can validate a design against more test vectors than using RTL model in the same amount of time.

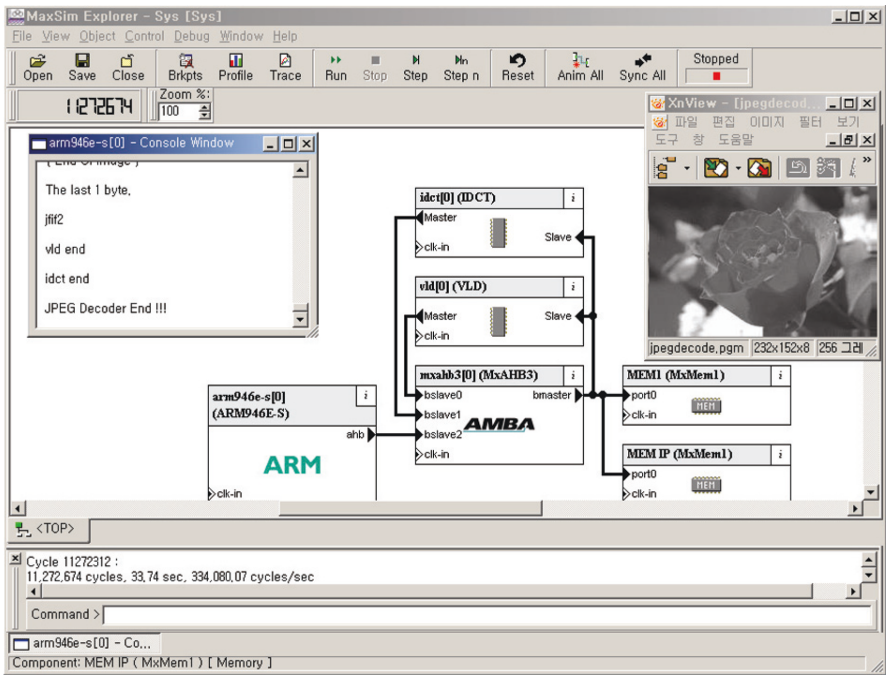
The debugging of TLM is much easier than that of RTL model because one can concentrate on the system-level operations without being harassed by excessive implementation details. Once verified, the TL model becomes a reference model for the following low-level implementations. Refer to Wieferink (2004) and Cai (2003) for further information. TLM typically includes adapters for converting transactions to/from transition activities of interface signals so that one can mix TL models with RTL models in a single simulation environment. This enables gradual refinement from TL model to RTL model as design evolves from high-level specification to low-level implementation.

AMBA AHB Cycle Level Interface (AHB CLI) Specification (ARM, 2003) sets a standard for modeling AMBA AHB bus in micro-architectural TLM with SystemC. The specification defines basic building blocks of AMBA bus: master, slave, arbiter, decoder and bus itself, and interfaces provided by them. The interface method call can be directly correlated to AHB signal events. In addition, the evaluation sequences of the blocks and data types used by the bus models are specified in the AHB CLI specification. System Studio from Synopsys and ConvergenSC from CoWare already support AHB CLI specification in the form of CLI-compliant ARM and AHB bus models in SystemC along with SystemC simulator kernels.

2.3 Case Study

Figure 3 shows an implementation of JPEG decoder in transaction-level modeling with SystemC. We used MaxSim of ARM to realize the JPEG system composed of a single ARM946 processor, Inverse Discrete Cosine Transfer (IDCT), Variable Length Decoder (VLD), two memory models, and a single AHB bus model. Transaction-level modeling makes it possible to interconnect building blocks in transaction-level rather than pin-level. Each building block, itself, is modeled in transaction-level; IDCT and VLD models are derived from algorithmic-level models and processor model and bus model are provided as a form of a simulation model with MaxSim. With pseudo cycle-accurate simulation model, we could achieve a simulation performance of 334kcycles/sec with JPEG decoder.

We can implement the same system with other SystemC-based simulation systems as well. These include ConvergenSC from CoWare, System Studio from Synopsys and OSCI reference SystemC simulator.



```
The last 1 byte.
jfif2
vld end
idct end
JPEG Decoder End !!!
```

```
Cycle 11272312 :
11,272,674 cycles, 33.74 sec, 334,080.07 cycles/sec
Command >
```

Component: MEM IP (MxMem1) [Memory]

Figure 3. Implementation of JPEG decoder with MaxSim®

3. HW-SW CO-VERIFICATION

In HW-SW co-verification, software that will be running on the target processor is simulated with hardware models that mimic the target hardware system. Software engineers usually develop software code and verify its functionalities in a host machine by compiling and running the target software code on the host machine, so-called native code execution, at the beginning. It is, however, not enough to develop hardware-dependent software such as boot loader or device drivers without hardware models. Through the HW-SW co-verification, software engineers are now able to develop and debug their software by simulating it at system level with the target hardware system even when prototypes are not available yet. Co-verification with the target hardware model also makes it possible to evaluate performance or estimate energy consumption of the target system in the early stage of the design process, which is essential for software optimization considering the target architecture. In other words, the software verification and optimization step that is traditionally followed on prototyping step in the design process is able to progress simultaneously with hardware verification step; as a result, design time is much reduced. HW-SW co-verification also benefits hardware engineers. Each hardware component is usually verified with synthetic HDL test-benches that are made by the hardware engineers at the beginning; it cannot provide comprehensive test coverage. Through the co-verification, hardware engineers are able to verify the hardware components with large stimulus that is much closer to real system.

Important metrics of co-verification are simulation speed, accuracy and visibility. HW-SW co-verification is accompanied by simulation performance degradation due to a number of simulation models and communication overhead between the models. Slow simulation speed is a problem especially to the software engineers. Sometimes, it is necessary to co-simulate the target system all day long for OS booting in order to find a software bug in an application. Accurate simulation is important for hardware engineers who design a hardware block while imagining signal transitions cycle by cycle. The most important feature of simulation that is much better than that of emulation (prototyping) is visibility, which is indispensable for both software and hardware engineers to debug their design. Besides, system profiling features, such as bus and cache monitoring, is useful to find critical path and optimize design.

This subchapter introduces processor modeling techniques, and then describes various combinations of simulators for co-verification.

3.1 Processor Simulation

A processor can be modeled as an Instruction Set Simulator (ISS) that fetches instructions from memory (or memory model) and simulates each instruction behavior sequentially. ISS is a widely used design and validation tool for both hardware and software engineers. It is useful to evaluate instruction set architectures (ISA's) during the architecture exploration and to validate the compiler, operating system, or application software when the actual silicon is not yet available. ISS is also used in HW-SW co-verification for processor models, that is, to execute target software. There are three methods in modeling a processor: interpretive ISS, static compiled ISS and dynamic compiled ISS.

3.1.1 Interpretive ISS

The basic method for the ISS is interpretation. The interpretive ISS executes an instruction simulation loop, "*fetch-decode-execute*," as having the state of the target processor in memory as shown in Figure 4, which is similar to the activities of single issue processors. Interpretive ISS is widely used due to easy implementation and high flexibility.

Typically interpretive ISS is two orders of magnitude slower than native execution where target code is compiled for host processor and run on the host machine, as several tens of host instructions are executed to simulate a single target instruction. As the time for instruction fetch and decode, line 2 and line 3 in Figure 4 consumes a large portion of simulation time, some researches attempt to speed up these steps using various techniques. Nohl et al. (2002) introduced so-called *just-in-time* cache, which is a simulation buffer saving decoded information of recently simulated instructions. There is no need to fetch and decode instructions that were used before and are still in the cache. Another reason for the low speed of interpretive ISS is that an interpretive simulation should perform some time-consuming operation that can be redundant owing to lack of the knowledge of future events. For instance, overflow calculations performed for every data processing instructions are not necessary if following instructions do not use it.

```
for ( ; ; ) {
   inst = fetch( PC );
   decode( inst, &opcode, &op1, &op2, &op3 );
   swtich( opcode ) {
      case ADD:
         op1 = op2 + op3;
         break;
      case MOV:
      ...
}}
```

Figure 4. Simulation loop of interpretive ISS

3.1.2 Static Compiled ISS

Static compiled ISS is much faster than the interpretive ISS because the fetch and decode steps are performed as a batch in the start-up procedure. The static compiled method translates the whole target program, which is an executable binary for the target machine, into the target simulation program running on the host machine. This means each target program will be a unique simulator.

There are two translation schemes: One is binary translation that is the direct replacement of target instructions by host instructions, while the other scheme goes through high-level language generation and compilation stages. Figure 5 shows those simulator generation schemes. If the target and host machine have the similar instruction architecture, the implementation of the former method, binary translation is relatively easy. Otherwise, the binary translation can be difficult and accurate simulation cannot be guaranteed because, in some cases, there is no way to simulate a target behavior with a combination of host instructions. Moreover, the binary translation does not have host compatibility. The latter method is easier to implement and can be applied independently of the host machine, since it uses C (high-level) language. The static compiled ISS that uses the C intermediate code is introduced by Zivojnovic et al. (1995). Another advantage is in simulation speed. The host C compiler removes the redundant operations through various optimization techniques, such as dead code elimination.

Although static compiled ISS is faster than interpretive ISS, almost all the commercially available ISS's are, however, interpretative because of the following limitations. First, the static compiled method has a considerable start-up cost due to the generation of simulation code and its compilation, which can be a major drawback for large software simulation. Even for the

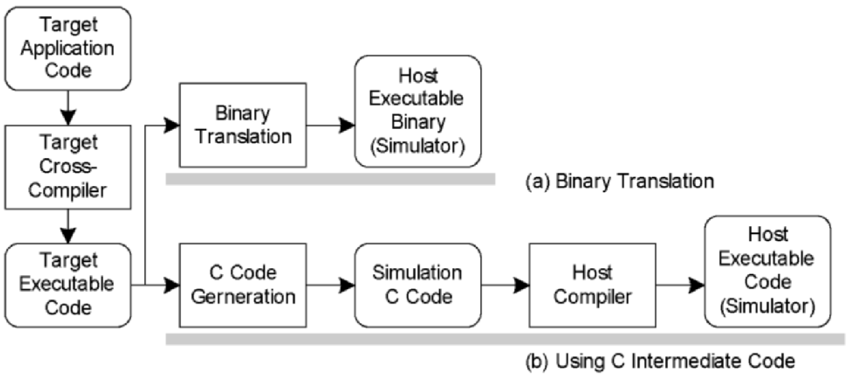


Figure 5. Static compiled ISS

partial modification of the target software, user should go through all the simulator generation processes. All instructions in the target binary code are translated into C simulation code one by one, so the simulation C code becomes a large, complex and unstructured C function with labels and '*goto*' statements for all branches. The compilation time of the C simulation code increases in a super-linear way with the size of the function. Therefore, it is necessary to divide the generated C simulation code into many suitable size functions to reduce the compile time. To do this, the unstructured C code should be translated into C code having structured control flow by resolving the destinations of each branch. Chung and Kyung (2004) proposed object-based compiled ISS. Object files holding symbol information are used to generate C simulation code instead of the binary, so it is possible to generate C code having the same function structure, and, as a result, the compilation time is increased. Incremental compilation is also possible since each source file is processed separately. Figure 6 shows the incremental compilation of compiled ISS.

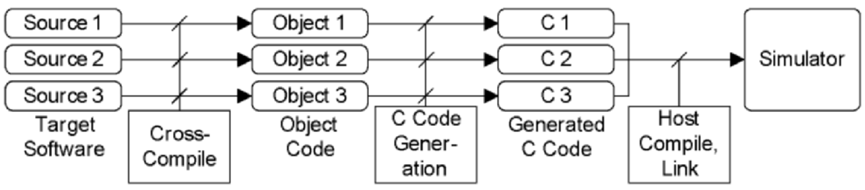


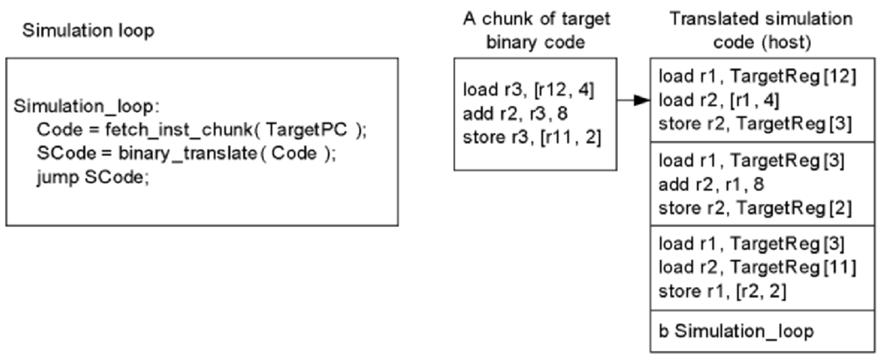
Figure 6. Compilation of object-based static compiled ISS

Second, the static compiled ISS has restrictions on flexibility. Since the static compiled method assumes that the complete program code is known before the simulation starts, it cannot support the dynamic code that is not predictable prior to the runtime. For example, external memory code, self-modifying code, and dynamic program code provided by operating systems or external devices cannot be addressed by the static compiled ISS.

3.1.3 Dynamic Compiled ISS

The dynamic compiled method moves the compilation process to run-time. Each chunk of the target binary is translated into host execution code on the fly (Cmelik et al., 1994). Since C compiler is not adaptable to compiling the chunk of the C code supplied dynamically, the binary translation method is usually taken. Figure 7 shows the main simulation loop and an example of binary translation (Witchel and Rosenblum, 1996).

Instruction fetch, decode and translation steps are time-consuming and can be reduced by cache in the same way as interpretive ISS. To maximize


Simulation loop
A chunk of target binary code
Translated simulation code (host)

```
Simulation_loop:
    Code = fetch_inst_chunk( TargetPC );
    SCode = binary_translate( Code );
    jump SCode;
```

```
load r3, [r12, 4]
add r2, r3, 8
store r3, [r11, 2]
```

```
load r1, TargetReg[12]
load r2, [r1, 4]
store r2, TargetReg[3]

load r1, TargetReg[3]
add r2, r1, 8
store r2, TargetReg[2]

load r1, TargetReg[3]
load r2, TargetReg[11]
store r1, [r2, 2]

b Simulation_loop
```

Figure 7. Dynamic compiled ISS and an example of binary translation

simulation speed, it is necessary to simulate target architecture directly utilizing the resource of host machine rather than simulating it. For example, in Figure 7, the registers of the target processor are modeled with array variable, i.e., *TargetReg[ ]*, in the host memory. However, some of the host registers can be allocated for simulating the target registers without host memory access. Zhu and Gajski (2002) aggressively utilize the host machine resources through register allocation API's in order to get the faster simulation speed.

3.1.4 Native Code Execution

Native code execution (or direct execution or host execution) is used for functional verification of software in the early stage of design process. Fast running speed and plenty of debugging tools are beneficial to software engineers. However, the simulation accuracy is poor because the host processor usually has different architecture from the target processor, e.g., different instructions, MMU, cache, etc. It supports only high-level language, such as C, where processor architecture is transparent to the user for the same reason. Even for the functionality verification with native code execution, the following should be considered. Bit width of data types should be identical. For instance, bit width of an integer variable can be 16 or 32, which depends on the bit width of data path. A program verified on the 32-bit host processor may cause overflow in 16 bit embedded processor. Floating point operations and endian of memory layout should be considered as the same way. Embedded processors may not have floating point unit and different endian from the host processor.

Delay annotation allows extracting the timing information of the target processor from native code execution (Bammi et al., 2000). '*delay(cycle)*' function accumulating time consumption (cycle) of the target processor is inserted after each C statement or basic block in the target application code. One can obtain the time consumption by analyzing the C source code or cross-compiled target executable at static time ahead of simulation. However, the accuracy is not guaranteed. It is difficult to find the exact time consumption due to target compiler optimization and instructions that consume a variable cycle count depending on its operands. Lee and Park (2003) annotate the delay in intermediate representation (IR) of portable host compiler to increase the accuracy of the time consumption and portability. Most optimization of compile process is performed before the machine code from IR.

Another problem of native code execution besides inaccuracy is how to hook input/output (IO) activities from/to external hardware components simulated by other simulator. This issue will be addressed in section 3.2.5. Table 1 summarizes the advantages and disadvantages of each processor modeling method.

Table1. Comparison of processor modeling methods

	Interpretive	Static compiled	Dynamic compiled	Native code execution	HDL model
Simulation speed (IPS)	1~10 M	10~100 M	10~100 M	> 1 G	10~100
Accuracy	Good	Good	Good	Worst	Best
Stat-up cost	Small	Large	Small	Small	Large
Flexibility	Good	Bad	Good	Good	Good
Debug ability	Good	Good	Good	Best	Bad
Simplicity	Good	Bad	Bad	Best	Worst
Co-simulation with other HDL simulator	Good	Good	Good	Bad	Best

3.1.5 Other Issues on ISS

To develop application-optimized processor or DSP cores rather than to bring in off-the-shelf cores often gives us the optimal system for a specific embedded application. In this case, one can evaluate the various alternatives of instruction set architectures by using ISS. To allow early evaluation of the architecture along with software optimization, on-the-fly generation of ISS or retargetable ISS has been introduced. One can add instructions and/or modify the behavior of instructions by simply modifying descriptions of instruction set architecture, which can automatically generate the corresponding ISS (Pees et al., 1999 and Schnarr et al., 2001.) Besides, automatic generation of assembly and C compiler and debugging tool for the generated ISS is quite effective to meet the time-to-market.

In addition to the core modeling, it is necessary to simulate other components in the data path, such as memory management/protection unit (MMU/MPU), cache, etc. One can simulate these components by simply appending their simulation models to the data path of the processor. However, it is not easy to model parallel activities of hardware by sequential software execution. Fast and accurate simulation of advanced processor architecture, such as out-of-order execution, superscalar or multithreaded architecture is an issue of ISS. Especially for the static compiled ISS having static execution sequence, it is a pending problem, since actual execution sequence is determined at run time to maximize the performance.

3.2 Co-Verification

Native code execution or ISS undertakes software simulation while a logic simulator performs hardware component simulation. For the HW-SW co-verification, either a simulator models both hardware and software, or the two simulators, for HW and SW respectively, work together while communicating with each other. This section introduces alternatives of simulator combinations for co-verification.

3.2.1 ISS with C Hardware Model

ISS mainly offers C API (Application Programming Interface) to support user C object working together with the processor model through dynamic linking library. In general, it is used for simulation of hardware stubs without hardware simulator or for profiling the application running on the ISS. There are many problems on modeling concurrent hardware with sequentially executed C language. In addition, the C hardware modeling can become a useless job, because the C model cannot be seamlessly translated into lower abstracted HDL code yet. For those reasons, ISS with C stubs as shown in Figure 8 is an inefficient way for system-level simulation, where all system components are simulated with real applications.

Although the C stubs are very abstracted and not proper for top-down design refinement process of hardware design, it is very useful for software engineers. Simple peripherals, such as timer or PIC (Programmable Interrupt Controller) can be easily modeled with C language and embedded in the processor model. Typically hardware components necessary for embedded software to run are simple peripherals, e.g., timer required for operating system or UART for serial communication. Consequently, software engineers can simulate and debug their software using C stubs for minimal hardware components. Software debugging has become easier since most processor vendors now provide ISS with various debugging features.

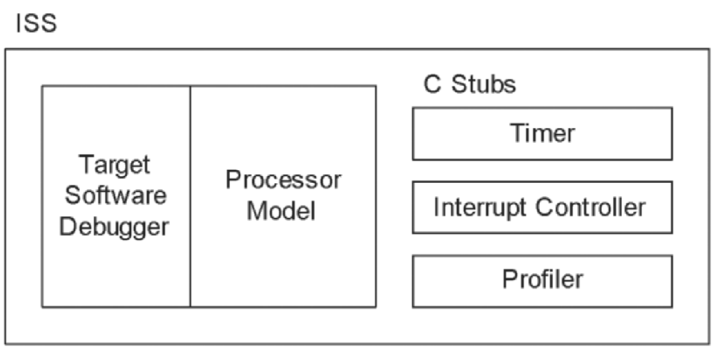


Figure 8. ISS with C stubs of hardware components

3.2.2 HDL Simulator with HDL Processor Model

It is also possible to co-simulate the target system only with an HDL logic simulator including both hardware components and processor modules, if the target processor logic is available. If the simulation of processor behavior is purely in logic level and the communication to the hardware components are bit-accurate, HDL logic simulation is the most accurate simulation method. However, the simulation speed is poor, typically several tens of instructions per second, which is almost useless for software development. Rather than system-level co-simulation, it is beneficial to hardware logic debugging including processor with short test-bench programs.

Instruction-level debugging is feasible through waveforms of processor registers and memory contents, but remote debugging tools are required for source-level debugging. Many software debuggers support remote debugging features when the software is not running on a host computer but on a remote machine whose resources (memory or storage devices) and/or user interfaces (keyboard or display) are not sufficient for self-debugging. Hence, the source-level debugging is possible using the remote debugger attached to the HDL processor model instead of the remote machine. Figure 9 shows the co-simulation with HDL processor model and remote debugging.

GUN Debugger (gdb) supports remote debugging features for various target processors. As all debugging features of gdb are based on software implementation, there is no need to make additional hardware logic for debugging such as scan chain. The only thing user has to do for the gdb connection is make a trap hander of the target processor responding to gdb commands, i.e., register/memory read/write through gdb remote protocol. Because these software-based debugging techniques execute additional

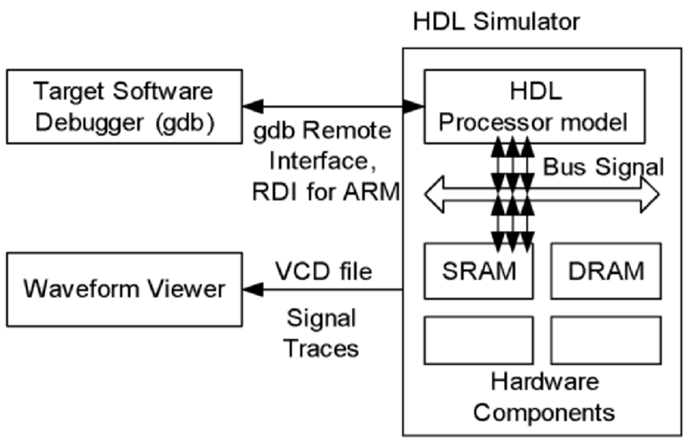


Figure 9. HDL simulator with HDL processor model

processor instructions and make bus transactions and memory accesses, which do not occur in normal mode, there are differences in hardware activities between the debugging and normal mode. Therefore, a fault in normal mode sometimes does not occur in the debugging mode, and vice versa, especially when the hardware is not stabilized yet.

3.2.3 ISS with HDL Simulator

ISS and HDL simulator combination is a popular co-simulation method many co-simulation tools support. The co-simulation tools take charge of synchronization and communication between the two existing simulators, so both software and hardware engineers can keep using familiar simulation environments for co-simulation. Since each simulator occupies a process of host operating system, IPC (Inter-Process Communication) is used for connecting them. There are two communication methods with different abstraction levels of the communicating data: one is transaction level, i.e., bus transaction (read/write) information without detailed signal information and the other is signal-level communication, transferring bus/pin signal values. In general, instruction-accurate ISS’s represent bus activities through transactions without detailed description on pin signal information. In this case, bus functional model (BFM) in the logic simulator translates bus transactions between transaction level and signal level. If the ISS is cycle-accurate and port activities are available, then the signal-level communication is more appropriate. Proxy module of the processor mimics the HDL processor module communicating with the ISS. Figure 10 shows both combinations.

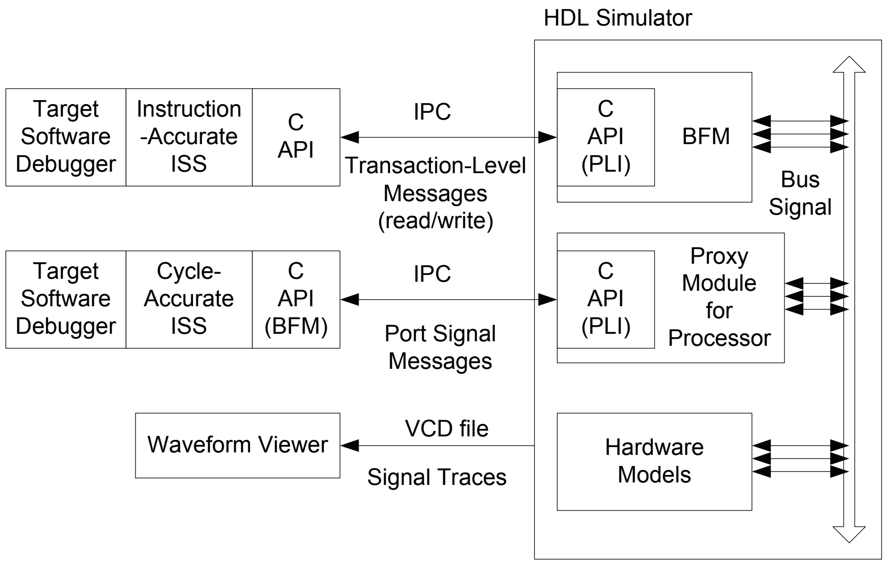


Figure 10. Instruction-accurate ISS and cycle-accurate ISS with HDL simulator

The slowness of the execution speed of the slower simulator, i.e., HDL simulator, and the time for synchronization and communication between two simulators are the bottlenecks in the performance of the co-simulation. Especially for the signal-level communication, the values of the port signals need to be shared every clock cycle; thereby the simulation performance is limited by the communication overhead. Now, co-simulation is also facing the same problem as the traditional simulators had to, i.e., trade off between accuracy and speed.

As mentioned above, debugging facilities for software and hardware component design are also available for the co-simulation. Many co-simulation tools offer various profiling features helpful for early design optimization, such as analyzing time-critical portion of software code, cache profiling, and bus monitoring, i.e., delay or contention profiling.

3.2.4 ISS with SystemC

SystemC is a C++ class library for hardware description and simulation, HDL simulator in the ISS-HDL simulator combination can be replaced by SystemC model (Benini et al., 2003). Processor class is a shell of processor model communicating ISS, and no PLI interfacing is needed since the SystemC model is based on C++ language. gdb commands can be used for interfacing two simulators through pipe without interface protocol setup as shown in Figure 11. An advantage of this combination is that SystemC is

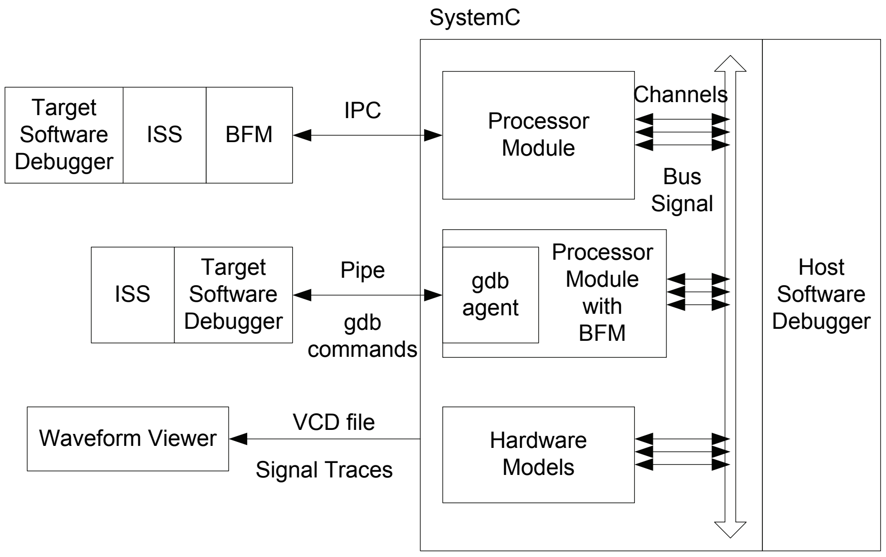


Figure 11. ISS with SystemC hardware models

better to describe hardware at higher level of abstraction, which is important to meet the time-to-market. In addition, gradual refinement to HDL code is possible since today's HDL simulators support simulating HDL-SystemC mixed design.

Co-simulation with a single simulator for both hardware and software allows faster simulation and seamless design flow. SystemC allows hardware engineers to use the C/C++ language for modeling hardware, and both hardware and software models can be simulated in a single host process with the SystemC simulation kernel. There are, however, difficulties in simulating the software running on the target processor, because SystemC does not have any processor models. Figure 12 shows solutions: one is embedding ISS in SystemC processor module, and an alternative is to use native code execution instead of the ISS.

3.2.5 Native Code Execution with SystemC

SystemC is a proper language for high-level design, so the combination of the native code execution and SystemC is most suitable for high-level co-simulation and design exploration with much faster simulation speed (Blaurock, 2004 and Chung et al., 2005). There are two problems for the combination: synchronization and communication. To synchronize with hardware clock events, native code execution should be stopped at certain points and wait for corresponding clock events of the hardware model. A solution is to annotate delay functions into the source code as mentioned

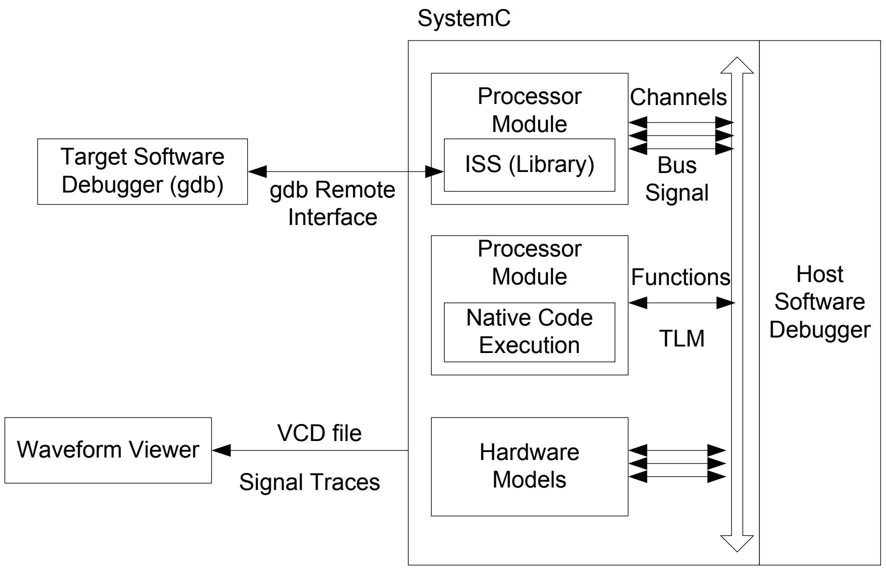


Figure 12. Embedded ISS and native code execution with SystemC

in section 3.1.4 of this chapter. Although the calculated cycle consumptions cannot be exact due to the target compiler optimization, it is generally enough for the purpose of high-level simulation.

To solve the communication problem, read and write operations of native code should be translated into bus transactions of the hardware model. There are three methods for this, i.e., how to hook IO access from native code execution. One is to replace IO access code with a function that accesses hardware component model. It can be done either when preprocessing through C code modification before compilation or during compile time through machine code replacement. Another method is to use a trap of the host machine. One can make the IO variable access initiate software trap, and the trap handler generates bus transactions to the appropriate hardware model and returns to the software execution flow with the obtained data to continue execution. The third method is to use operator overloading of C++ language, which allows a class to rebuild the behavior of operators related to the class instance. One can make an IO variable class, such that read/write from/to the IO variable class initiates bus transactions of SystemC hardware model using the operator overloading.

3.2.6 Heterogeneous Simulation Environment

System components of today's complex SoC design can be modeled with various languages and run on various simulators, such as ISS, HDL simulator, SystemC, hardware-based simulator and prototype board. In order to validate the design at system level, more than two simulators often work together with their clocks synchronized and pin signals shared. In this heterogeneous simulation environment, synchronization and communication overhead between the simulators degrades simulation performance, which becomes one of the most important issues in the system-level simulation of SoC as the complexity of SoC increases as it should.

4. HARD PROTOTYPING

This section covers general issues on hard prototyping. By the term 'hard prototyping', we emphasize the prototyping systems actually utilizing the hardware components. However, in this chapter we extend the meaning of the hard prototyping to include the software part integrated into one unified environment.

We will first summarize conventional hardware-assisted verification tools for logic-centric LSI chips. Then, additional requirements for the SoC verification will be explained. General issues of the hard prototyping will then be addressed with some highlight on the debugging issues. The efforts on standardizing emulation systems will also be addressed.

4.1 Classification of Hard Prototyping

The three major categories of classical verification techniques assisted with hardware equipments are hardware acceleration, hardware emulation and prototyping as shown in Figure 13. References can be found in Keating (1999), Rashinkar (2001) and Staunstrup (1997). All three verification techniques are utilizing hardware equipments as a vehicle to run all or part of the DUV (Design Under Verification). However, the focus of verification is slightly different among them.

Although hard prototyping is classified in this section, it is hard to strictly differentiate among these categories. For example, the acceleration equipments can be used for emulation if the speed is fast enough and the external connection is available to interface with existing development boards. The emulation equipment can also be used as a prototyping system if it is small enough and cost-effective. Especially for SoC development and verification, the boundary between emulation and prototyping is ever smearing, since software takes ever more portion in SoC.

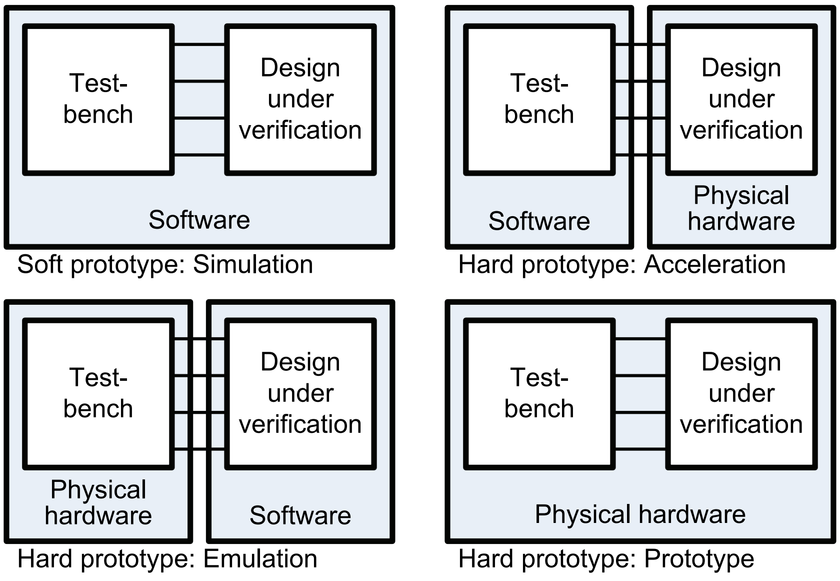


Figure 13. Classification of hard prototyping

4.1.1 Acceleration

The acceleration, which usually is an abbreviation of the simulation acceleration, is focusing on increasing the speed of simulation while keeping the environment as similar as possible to the simulation environment. Originally, it was invented to accelerate time-consuming simulation jobs such as gate-level simulation or fault simulation as explained in Eiriksson (1990). However, as the design size increases with enhanced process technologies, it becomes useful also for the RTL or higher-level logic simulation. One extreme application area of acceleration technique is to accelerate the simulation of algorithmic models or architectural models described in MATLAB.

For the acceleration of RTL logic simulation, maintaining the same test bench structure as one used for the simulation and providing higher visibility are very important.

4.1.2 Emulation

The emulation is used to verify the functionality of complex DUT (Design Under Test) using specialized hardware equipments in the context of realistic operation environment. Usually massive array of FPGA is used to accommodate the functionality of the DUT, but specialized processor array is also adopted in nowadays high-end emulation systems. The main focus of verification is on the behavior or functionality of the DUT itself. Therefore, the emulation technique is often used in conjunction with existing test board slowed down to meet the speed of emulation equipment. In such a case, the emulation equipment provides a way to mechanically plug itself into the existing board, so it is called 'in-circuit' emulator. The confidence level of the emulation is higher than that of the acceleration in the sense that more realistic test environment is used for the test and longer verification scenario is applied to the DUT.

4.1.3 Prototyping

The prototyping is used to verify and demonstrate the functionality of the whole system including the chip. It is concentrating more on the software-side. After the chip design is nearly completed, and before the chip is taped-out, the prototyping enables the software engineers to start software development and/or verification. At this stage, therefore, not only the behavior of the chip, but also the operation of the whole board should be quite near the final product.

Prototyping is more than simple chip design, i.e., prototyping realizes a concept or a design rapidly prior to the final production or mass production, to see the functionality, visualize the shapes, estimate the cost, or analyze the potential defects. Sometimes the form factor or physical dimension matters more in prototyping compared to the emulation which is more focused on functional aspects. In addition, the cost-effectiveness is emphasized more in the prototyping than in the emulation.

4.1.4 Requirements of hard prototyping

The requirements for the hard prototyping can be summarized as follows.

- Gate-capacity: the gate capacity of the equipment must be sufficient to contain the whole design. Usually hard prototyping equipments using large number of FPGA require bigger gate capacity than actual design size because of the overhead incurred by the inter-FPGA partitioning.
- Speed: the operating clock frequency of the hard prototyping equipment must be high enough to run the application code with a sufficient time margin. Often, typically for emulation, the target system needs to be slowed down to meet the performance of the emulation equipments.
- Configuration time: in most hard prototyping equipments it is needed to configure all the programmable devices to work as a part of user design,

before starting execution. This configuration time should be kept short enough compared to the actual running time of the simulation/emulation.
- Compile time: a fair amount of time is spent to prepare the configuration data for the programmable devices. Typically this includes partitioning time, synthesis time, mapping time and placement and routing time. All of these components are linearly or exponentially proportional to the number of the programmable devices used. Therefore, there is a trade-off in choosing the number of programmable devices, between gate capacity and compile time. Parallel or distributed compilation techniques and incremental compilation techniques can be applied to enhance the compile time.
- Visibility: for the debugging of the user logic, it is essential to provide a way to record and show the events internal to the programmable devices. Generally increasing the visibility will cause negative effects on other metrics such as the gate-capacity, the operation speed, the compilation time etc.

Other issues which need to be considered include extensibility, scalability, maintainability, ease of use, cost and so on.

4.1.5 Examples of conventional hard prototyping system

Many EDA vendors are eager to have their own tool chain starting from conceptual design seamlessly connected to the hard prototyping. Commercial products are typically focused on acceleration and emulation.

The most well-known FPGA based emulator is System Realizer family from Quickturn, part of Cadence Design Systems. It used massive array of FPGAs with the custom emulation software that enables distributed compilation for the FPGA and debugging. Quickturn also had a custom processor-based emulation system, CoBALT and Palladium which is now a major emulator product line of Cadence. These systems use an array of custom processors that can be configured to the end-user design with compilation. Each custom-processor approach shows a slightly slower speed than FPGA but has more flexibility, so that the processor-based emulators can be used as a simulation accelerator. Mentor Graphics also has a custom processor-based emulation system family, Celaro.

Axis (Cadence) used hybrid approach that utilizes FPGA, but instead of directly mapping user design into FPGA, maps synthesizable processor core called RCC (Re-Configurable Computing) to the FPGA and executes compiled codes of the user design in the processor.

The prototyping is usually done by the end user or as a technical service due to its dependency on the final product. One noticeable exception can be found from Aptix. The System Explorer series is a prototyping platform providing flexible interconnection network using their own custom interconnection chip, FPIC (Field-programmable interconnect chip) among several PCB modules, some of them pre-designed while others customer-provided. The pre-designed modules cover FPGA modules mounting recent devices from major FPGA vendors and system modules mounting microprocessors, DSPs, interface circuits, analog circuits, etc.

There is an extreme case of emulation that utilizes behavioral-level model and connects it to the real target board (Kim et al., 1998 and Dynalith, 2000). The behavioral-level model is usually written in the C language. With this approach, SoC chip model in algorithmic representation can be verified along with actual target board.

4.2 Evolution Toward SoC Verification

For the verification of SoC, further requirements need to be provided by the acceleration, emulation, and prototyping equipment.

4.2.1 Processor modeling/integration

For the verification of SoC, the hard prototyping system must be able to handle the target processor used in the SoC. One way is to use the behavioral model for the processor. The other way is to attach actual processor implemented in discrete component to the hard prototyping system (Dynalith, 2004a). The former which is discussed in section 3 of this chapter, is rather essential and easier for debugging, but slower than the latter. In addition, the latter is only applicable when the compatible discrete chip is available and also its modeling accuracy will be limited by the I/O interfaces and the bus structure of the discrete chip.

4.2.2 Large memory modeling/emulation

Large memory elements are used in SoC design for several purposes such as code memory, data memory, FIFO memory for inter-module data communication, etc. The most straightforward way to support them is to provide many discrete memory components having the same or larger size than required. These discrete memory components are useful for not only modeling the memory internal to the SoC, but also providing off-chip storage in final SoC system. Unfortunately, the type and size of the discrete memory components are so various that it is impossible to adopt all kinds of memory devices.

One way to overcome this is to provide a generic memory wrapper model to mimic the interface of specific memory standard (Gharsalli, 2002). With this scheme, one large physical memory device can be shared among several logical memory devices of different types.

Another general way to solve the memory problem is to use software simulation model for the memory. This will be slightly slower but provides better visibility and flexibility

4.2.3 Integration with simulator

As the complexity of SoC and its corresponding test bench increase, it becomes necessary to run the hard prototyping environment in cooperation with the software simulator. Its purpose was mainly to accelerate simulation by running already verified IP's in the hardware and running only part of the design under verification in the simulator with enhanced visibility of the software simulation.

Nowadays, it becomes more popular even in the emulation or prototyping systems, to run all or part of the design in the hard prototyping system and connect them with the simulator to use the enhanced features of the simulators for the test bench modeling, system modeling, coverage analysis and formal techniques.

4.2.4 Co-work with behavioral-level model

In SoC design and verification, ever-increasing is the need for the continuous design flow from the high abstraction-level modeling to the gate-level implementation. For this, it is required to interconnect the high abstraction-level model written in various languages to the hard prototyping system.

The purpose and usage of the behavioral model varies depending on the verification methodologies. For example, the behavioral model for the part of the design running in the hard prototyping equipments can also be run for the generation of test results to be compared with the output of the DUT. Or they can be used as models for not-yet implemented DUT modules.

Usually, the behavioral models are lacking detailed timing information. Therefore, some additional efforts are required to bridge the 'un-timed' or 'roughly-timed' behavioral model to the 'full-timed' or 'exact-timed' models such as hardware equipments.

4.2.5 Case study 1: ARM-based prototyping solution - ARM RealView Integrator and Versatile

ARM RealView Integrator and Versatile is the typical prototyping system where designers can evaluate the target microprocessor within the real hardware environment incorporating configurable bus and peripherals such as UART, LCD, memory, Ethernet and so on (ARM, 2004, 2005).

In this platform, software and hardware designers can work together to integrate and emulate the whole design. The software designer can compile her program and download executable code into its memory to run and test application programs. The hardware designer can build various AMBA bus system architectures in FPGA. In addition, real hardware peripheral IO even increases the confidence of the verification.

The PCB board basically can be expanded by installing daughter boards through stacking interconnection. The designer can simply use various ARM core families by exchanging ARM core daughter board and expand configurable logic by installing additional FPGA daughter board.

4.2.6 Case study 2: Bridging emulator to the test bench in iPROVE

As a typical example of the hard prototyping system for the SoC verification, iPROVE from Dynalith Systems (2002) provides enhanced communication channel to the host computer in various abstraction-level mode. iPROVE is a PCI-based single-FPGA emulation/acceleration solution for medium scale ASIC SoC design. By adopting only one FPGA, iPROVE can still cover most practical user designs, either IP or the whole chip, as the gate capacity of a single FPGA exceeds several tens of million gates, and it eliminates needs for partitioning user design and minimizes the compile time overhead. However, if user design exceeds the capacity of the FPGA, it still can be applied for partial design verification, which is a compromise between the speed and gate capacity of the verification model.

iPROVE provides three different types of operation mode for co-working with host computer, i.e., running HDL simulators, instruction set simulators or behavioral model in high-level language.

- Cycle-level mode: in this mode, the design in the FPGA and the HDL simulator in the host computer can communicate in clock-cycle-accurate manner. In addition to HDL, SystemC (Ki et al., 2003) or pure C/C++ can be used as a part of system model. Various coverage tools or test automation tools such as SpecMan, e, Vera or TestBuilder can be used along with the HDL simulator greatly enhancing the verification quality.
- Transaction-level mode: in this mode, the design in the FPGA communicates with the test bench or behavior model in the host computer using FIFO style data channel. It requires a special hardware called ‘transactor’ to be designed to adapt the interface of the user design to the FIFO. Once it is designed, user can achieve several orders of magnitude faster operation speeds than in cycle-level mode. Also it can be re-used for design with similar interface protocol.

- Abstract bus mode: this mode is a special case of the transaction-level mode. In SoC design, a few bus standards are dominating all the interface standards. For such a standard bus, pre-provided ‘transactor’ hardware and corresponding API functions accessing it can be prepared to help user building the behavioral model in high-level language or connecting to the instruction set simulator for the processor. iPROVE is provided with the abstract bus mode for the AMBA bus system, which helps rapid build-up and verification of ARM processor-based SoC design (Dynalith, 2004b).

4.3 Issues on Hardware/Software Co-emulation

4.3.1 Synchronization

In order to perform two different levels of designs (e.g. C and Verilog design) working together, there are two methods: hardware/software co-simulation and hardware/software co-emulation. These two methods have the same capability of performing hardware and software models at the same time. However, they are different in that hardware/software co-simulation model is running on fully-software environment. On the contrary, co-emulation method incorporates not only processor but also hardware emulator. The processor takes care of the software model while the hardware emulator performs hardware model.

In most cases, co-simulation is referred to the system incorporating instruction set simulator (ISS) for software side and HDL simulator for hardware side. While software side of co-emulation could contain ISS, HDL simulator, native C code or any other software model, while the hardware side of co-emulation is composed of hardware accelerator or emulation incorporating FPGA, real-chip or real target board.

The hardware and software co-emulation method splits the coupled models into software and hardware sides. To let these models work together, a means of communication between them is necessary. In co-emulation, the communication between hardware and software is called *synchronization*. Through the synchronization process, heterogeneous models in different processing engines can communicate with each other and run together.

In this section, several levels of communication technique for communication time reduction are explained.

4.3.2 Level of communication

The communication in co-emulation system can be classified according to the level of communication. Here, the “level” means the “level of abstraction of communication data." There are several factors that could specify the level of abstraction.

- Abstraction in terms of time
- Abstraction in terms of data

According to the above factors, there can be several levels of communication. Two levels of abstractions are commonly exploited; cycle-level and transaction-level communication.

4.3.2.1 Cycle-level

The cycle-level communication is clock-cycle-accurate in terms of time and pin-signal-accurate in terms of data. That means that every single bit of signals is transferred at every clock cycles through co-emulation interface. For example, when hardware and software are interconnected through AMBA AHB interface, HADDR, HDATA, HSEL, HWRITE and all other AHB signals are transferred at every HCLK clock cycle.

As the cycle-level communication is cycle-accurate and pin-accurate it requires the communication time of sending all the pin signal values at every clock cycle: There is no 'free lunch' as we all know.

4.3.2.2 Transaction-level

The transaction-level communication is higher-abstraction level than cycle-level communication in terms of time and data. The communication data is just composed of the essential information for communication. In the example of AMBA AHB interface, we just transfer only address value, data value and type of transfer such as read or write. The basic unit of meaningful essential information is referred to *transaction*. The one *transaction* may have the information for more than one cycle. Thus, the communication needs not to be performed at every clock cycle in transaction-level communication.

The transaction-level communication is effective to reduce the communication time. However, it might have some mismatches in terms of time or data due to the high-abstraction communication. For example, when the bus error occurs during the burst transfer of AHB, we can detect the error event after the end of the bus transaction.

4.3.3 Communication time reduction

There are several purposes of the hardware and software co-emulation. The first purpose is to confirm the design by actually executing design in real hardware in connection with the associated embedded software. Another purpose is the acceleration of verification speed. Moving some of design into real hardware emulator can reduce the simulation overall execution time while the communication overhead offsets the benefits of hardware emulation. Therefore, to achieve the second purpose, it is necessary to reduce the communication overhead. In this sub-section, several such techniques are explained.

4.3.3.1 Reducing the amount of communication

The simplest and most straightforward method to reduce the communication time is to reduce the amount of communication data. To reduce the communication data, we can apply the following techniques.

- Raising abstraction level
  We can reduce the amount of communication by sending only essential information for communication. In this method, communication is done by transferring a series of commands containing type and data rather than all of the signals (Bauer, 1999).
- Using value change detection

In traditional cycle-level interface, it sends all of the pin signal values at every clock cycle. But not all of the signals are always changed at every cycle. Thus we can send only changed signal values to reduce the amount of communication. Although the concept of the method is simple, there are some issues to solve in its implementation. To indicate whether a signal is changed or not, we need to send the changed flag first before sending the actual pin signal values. But sending flags increase the size of communication data. To reduce the flag overhead, we can group some signals to share a single flag. Using protocol awareness, we can make a more efficient group (Ki and Kim, 2005).

4.3.3.2 Storing stimuli patterns for fast regression test

Another method stores input port data in the memory located in the emulator. When we perform additional simulation, system applies the pre-stored patterns to DUT and compares the outputs with the expected values. This method can remove communication overhead by not interacting with long test-bench. Although this method can be useful for fast regression test, we have to perform co-simulation at least once to get input port values and the expected results. Moreover, it can not be applied to the design which is self-driven or one that has non-deterministic behavior.

4.3.3.3 Partitioning in terms of communication efficiency

Most commonly, the partitioning criteria between software and hardware in co-emulation system is whether the design can be synthesizable to be applicable to the hardware emulator. However, these criteria might be inefficient in communication time point of view. Using the technique converting any code into synthesizable one, we can be free from the synthesizability limitation when we partition the hardware and software in co-emulation system, which can bring more communication-efficient partition of co-emulation system (Bauer, 1998).

4.3.3.4 Utilizing channel characteristics

This method is used to reduce the communication time while maintaining cycle accuracy. Instead of modifying the communication data, it utilizes the channel characteristics. Most communication channels can achieve high data bandwidth in burst data transfer while it is inefficient in single or small data transfer. But the cycle-level communication method inherently exchanges input primary port value and output primary port value at every clock cycle, which is not communication-efficient because of the size limitation of the data that can be exchanged in a single transfer, i.e., the bit-width of input and output primary port limit the burst size of communication. To increase the burst size of the data transfer, this method moves some part of test-bench into the hardware emulator to remove the data dependency between the output port and input port within test-bench. Without data dependency, test-bench can apply a large amount of input port values corresponding to many clock cycles without receiving output port value from the emulator (Kim et al., 2004).

4.4 General Issues in Hard Prototyping

This section covers general issues in hard prototyping ranging from the physical problems such as clocks and routing, to the architectural issues.

4.4.1 Processor-based vs. FPGA-based

Most hard prototyping systems are implemented using FPGA. FPGA are usually used for rapid prototyping purpose but they are also useful for end-product in small-size market. FPGA have a massive array of configurable logic cells which are very useful in emulating behavior of user design logics. FPGA also have configurable I/O cells arranged along the four sides of the chip. They can be used in various ways in interconnecting FPGA with external peripheral devices. Mapping user logic directly to the FPGA cells has limitation in debugging.

Hard prototyping system may be implemented using special purpose processors. In this case, each processor simulates the behavior of some part of user design, i.e. there is no one-to-one relation between user logic and the simulating processor. There is a great flexibility in the size of design that can be emulated. Code debugging can be done at the same time with the behavior emulation of original logic. The drawback of the processor-based hard prototyping is extremely high costs.

4.4.2 Clocks and global resources

In FPGA-based hard prototyping systems, each functional module in user design is one-to-one mapped to the logic cells of the FPGA. For the clock lines, the global resources should be used to deliver the clock signals to all the sequential logic elements throughout the whole FPGA with minimum skew and delay. Because the clock resources are limited in number and some of them are occupied by the system operation, users have to carefully utilize the clock resources. This is applied also for other global signals such as 'reset'.

The problem is more difficult when using multiple FPGAs. If a clock domain sourcing one clock signal is partitioned into different FPGAs, the clock source should be distributed to the FPGA with minimum skew so that the set-up and hold time constraints for all the flip-flops spanned in multiple FPGAs can be met. The set-up time constraint can be relaxed by slowing down the operation speed, but the hold time constraint can not be satisfied without controlling the phase of the skewed clock or inserting special holding logic.

In processor-based hard prototyping systems, this kind of physical problems can be avoided by mimicking the behavior of simulator using specialized processor instead of implementing the circuit in hardware.

4.4.3 I/O types

FPGA provides various types of I/O technologies in a configurable way. However, this flexibility is reduced while the FPGA is mounted in the PCB in a specific circuit configuration. This is one of the biggest obstacles in making universal verification equipment. When developing or using the hard prototyping system along with external hardware components, users have to survey the I/O types of the external components and its pin number carefully.

4.4.4 Partitioning & routing

In both processor-based and FPGA-based hard prototyping, partitioning the given user design into several pieces is a very time-consuming but important step. Ill-partitioned design will consume more area and routing resources and take more time to synchronize data between partitions.

Partitioning can be done in gate-level or in RTL. Classical emulation or acceleration equipments usually partitions design in gate-level. However, recent hard prototyping systems support RTL partitioning to enhance the debugging feature. RTL partitioning allows the hierarchical structure of the design be used as a guidance to partition the design, which also helps re-constructing the waveform in user-readable form than in gate-level.

After partitioning, the routing phase follows. The routing may be statically fixed in PCB or programmable using special hardware resources. One example of programmable routing resource is a special chip called field-programmable interconnection IC (FPIC). In statically routed emulation hardware, the routing is done by allocating proper PAD location for each FPGA. The routing flexibility is limited while the physical characteristics of the routing channel are better than using programmable routing hardware.

A frequently used technique when the number of routing resource is smaller than required, is to multiplex multiple signals in one FPGA, transmit using one physical signal line, and de-multiplex it in the other FPGA.

4.4.5 FPGA-dependency

Each FPGA family provides various advanced features for managing clocks, high-speed interface, and specialized hardware for multipliers and DSP circuits, etc. When using FPGA for hard prototyping, the selection of FPGA model significantly affects the final features of the hard prototyping system. For example, the Excalibur from Altera integrates ARM9 processor core and several configurable peripheral IP's with programmable logic devices. It is a good candidate for ARM-based SoC modeling, although it has limitation that its AHB bus architecture is fixed by hardware. Xilinx also provides PPC cores with Viretex-2pro and Virtex-4 family.

4.5 Debugging Issues

Debugging is one of the most important issues in hard prototyping. In classical logic-centric chip design, the main debugging target was logic and timing problem. The timing bug is said to exist in a circuit that does not satisfy timing constraint given to the combinational logic path to guarantee proper operation in a specific operating frequency range. The timing bug can be detected by timing simulation which is a logic simulation considering the cell delays and routing delays of the chip. This can be done only in processor-based simulation acceleration.

The logical bug is typically related with the logical flaw caused by illegal initialization, bit-width mishandling, typing error, wrong Boolean equation or conditional statements, unexpected synthesis tool behavior, etc and undetected in simulation.

Logic analyzer is the most essential debugging equipment for logic and timing debugging. Most logic analyzers provide high speed data link with hard prototyping equipment in probe type or connector type. The operating frequency of the contemporary logic analyzers is very high and various triggering modes and large amount of memory for data capture are provided, which enables detailed timing analysis on off-chip signal. However, logic analyzers require physical connection with debug target and signals internal to the chip should be extracted to the external pads for debugging, which is very tedious and bug-prone.

FPGA vendors provide built-in logic analyzer features utilizing unused internal logic and memory resources. It simplifies debugging by eliminating messy coupling with logic analyzer, but its capacity is limited by the amount of spare resources. Many hard prototyping equipments have extra hardware resources dedicated for logic debugging.

The software debugging in SoC verification highly depends on the processor type used and the debugging features provided with the specific processor model. Most embedded processor vendors promote the debugging hardware and debugging tools specific to the processor. For example ARM processor is provided with the embedded trace module that enables the trace of instruction and data processed in the processor. In SoC verification, a means to correlate the trace data generated by the processor debugging tool and the signal dump gathered for the hardware is necessary.

4.6 Standard of Co-Emulation Modeling

4.6.1 Background

The benefit of hardware acceleration/emulation is the reduction of verification time, which would in turn help meet the ever-shrinking time-to-market requirements. Hardware-software co-emulation technique is used when only some part of design could be applicable to hardware acceleration/emulation while the rest of design remains in software part. In this configuration of design, software and hardware parts co-exist and interact with each other to realize the whole design functionality. Verification of this kind of design needs specialized platform which mainly incorporates two parts, in which host processor executes the software-side design and hardware accelerator/emulator takes care of hardware-side design. This verification platform is called co-emulation system and the target design under co-emulation system is considered as co-emulation modeling.

Co-emulation system, on the other hand, requires much time and effort to develop the whole co-emulation platform. It is, therefore, not efficient to develop a new co-emulation platform whenever we need to verify a new design. In most cases, designers utilize the commercial 3rd party co-emulation system or reuse the existing customized platform.

The co-emulation system is composed of not only a main processing engine (host processor and accelerator/emulation) but also implementation-specific components such as communication channel between processor and accelerator/emulator, device driver, API, bus interface and so on. These components can be different among particular system implementations according to emulation system vendor and model.

This implementation-specific feature affects co-emulation modeling. Accelerator/emulator vendors have proprietary APIs. Hardware-side of design under verification is required to be modified to interface with platform-dependent communication channel and bus interface.

Verification engineers need to learn these platform-dependent API and hardware interface to model and implement his/her design on the specific co-emulation platform. Sometimes, we need to change co-emulation platform to satisfy the special requirement. Then, re-modeling of the previous co-emulation model is needed due to the mismatches of API and hardware interface between the previous and current co-emulation system.

Co-emulation platform itself has implementation-specific portions, which is totally dispensable to the verification engineer. One does not have to understand the details of implementation of co-emulation system but just need to verify one's design on this platform. To hide the implementation-specific things, co-emulation system should be based on the layered architecture. Moreover, to avoid the re-modeling efforts when possible change of co-emulation system, interface between the layers need to be defined well. According to these needs of EDA industry, Accellera announced Standard Co-Emulation Modeling Interface (SCE-MI) (Accellera 2003).

4.6.2 SCE-MI layered architecture

Co-emulation system based on SCE-MI also incorporates two processing engines, which are processor and hardware accelerator/emulator. To hide the implementation-dependent details and increase productivity from design reuse, it has a layered architecture as shown in Figure 14. Moreover, to reduce communication overhead between processor and hardware emulator, communication is done in message-based transfer rather than cycle-accurate signal level. Thus, transactor is located in the hardware emulator where message is decoded and resolved into cycle-accurate signals.

The architecture has four layers. Each layer performs a well-defined function. We will discuss each layer of the architecture in turn, starting from the top layer. From the application layer view point, test-bench is directly connected to DUT, but the test-bench is usually described in test case-oriented un-timed model ignoring detailed signal protocol. For example, when DUT is a kind of memory device, test-bench can just care about which data to store in memory without concerning about detailed memory interface protocol signals. DUT is described in cycle-accurate signal-level model such as RTL (Register-Transfer Level) model. The protocol layer is responsible for abstraction-level conversion. Socket creates a message from the stimuli of a test-bench. The transactor decodes the message which in turn is resolved into the detailed cycle-accurate signals. The infrastructure layer is responsible for transferring messages. Messages are transferred through several independent logical channels, each of which connects a software API with an associated port macro. The physical layer coordinates the functions required to transmit a bit stream over a physical medium such as PCI bus.

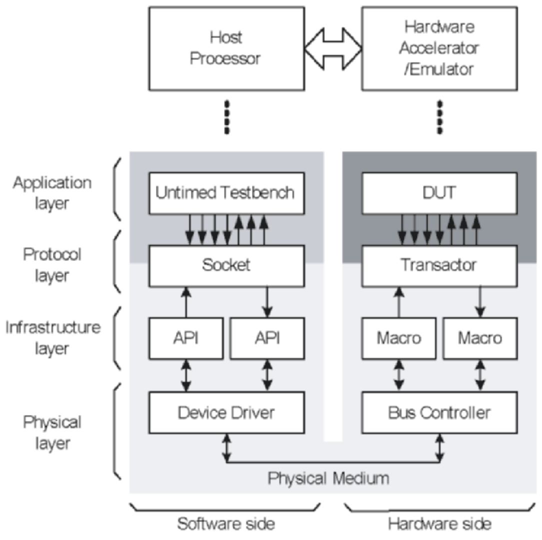


Figure 14. Layered architecture of SCE-MI

In this layered architecture, user can design transactor without considering the detailed emulator implementations. Emulation user can simply link between transactor and software test-bench through the standard transactor interface without the knowledge of emulation-dependent interface protocol.

4.6.3 Automatic Generation of Co-Emulation Interface

In the SCE-MI approach, emulator users don't have to be concerned about the emulator-system-dependent things using automated process which generate two bottom layers shown in Figure 14. Designs in protocol layer should be provided by the designer of protocol which is used for DUT interface. SCE-MI defines API function prototypes for software test-bench and protocol definition of macro module for standard transactor interface.

However, as these are just wrappers, we still need to complete the actual design of API and hardware macro.

To perform co-emulation in this environment, emulator user has to prepare bridge netlist, interconnects DUT, transactor and standard macro modules in RTL description using Verilog or VHDL.

Through the instantiated standard macros, user can connect transactor to the software test bench without knowledge of the detailed implementation of emulation system, i.e., users don't need to know the operation of emulation-dependent interface protocol for interconnection between processor and emulator. Actual interconnection is done by the automated processes as described in Dynalith Systems SCE-MI package (Dynalith, 2003). Through automated infrastructure linking process, emulator user can perform co-emulation modeling without concerning about implementation-dependent details and remodeling efforts in possible emulator change.

5. SUMMARY

In this chapter, SoC design flow and environment was addressed in terms of functional verification. In early design phase, soft prototype provides a working system model, where soft prototype consists of components models written in software including HDL, C/C++, SystemC and so on. With soft prototype, the technique of raising abstract-level is used in order to get reasonable simulation speed. One of the upcoming techniques is TLM based on SystemC. As one of prominent aspects of SoC is embedding processing cores, HW-SW co-simulation is an inevitable feature, where ISS simulates application software supposed to be run by the embedded processor. Several techniques are available for ISS, which include interpretive, static-compiled and dynamic-compiled. While design progresses, some or whole parts of SoC are replaced with hardware since the soft prototype can fail to deliver enough simulation performance. Hardware assisted techniques are called hard prototype comparing to the soft prototype. Hard prototype includes acceleration, emulation and prototype.

REFERENCES

Accellera, 2003, Standard co-emulation modeling interface reference manual, version 1.0; http://www.eda.org/itc.

ARM, 2002, ARM System-Level Modeling; http://www.arm.com.

ARM, 2003, AMBA AHB Cycle Level Interface (AHB CLI) Specification; http://www.arm.com.

ARM, 2004, ARM RealView Versatile Family Flyer; http://www.arm.com.

ARM, 2005, ARM RealView Integrator Family Flyer; http://www.arm.com.

Bammi, J. R., Harcourt, E., Kruijtzer, W., Lavagno, L. and Lazarescu, M. T., 2000, Software performance estimation strategies in a system-level design tool, Proceedings of International Workshop on Hardware/Software Codesign, pp. 82-86.

Bauer, M., Echer, E., Henftling, R., and Zinn, A., 1999, A method for accelerating test environments, EUROMICRO Conference.

Bauer, J., Bershteyn, M., Kaplan, I., and Vyedin, P., 1998, A reconfigurable logic machine for fast event-driven simulation, Design Automation Conference (DAC).

Benini, L., Bertozzi, D., Bruni, D., Drago, N., Fummi, F. and Poncino, M., 2003, SystemC cosimulation and emulation of multiprocessor SoC designs, IEEE Computer, 36(4):53-59

Blaurock, O., 2004, A systemc-based modular design and verification framework for C-model reuse in a HW/SW-codesign flow, Proceedings of International Conference on Distributed Computing Systems Workshops, pp. 838-843.

Cai, L. and Gajski, D., 2003, Transaction Level Modeling: An Overview. In Proc. IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'03), Newport Beach CA USA, pp. 19-24.

Caldari, M., Conti, M., Coppolar, M., Curaba, S., Pieralisi, L., and Turchetti, 2003, Transaction-level models for AMBA bus architecture using SystemC 2.0, In Proc. Design, Automation and Test in Europe and Exhibition (DATE'03), Munich Germany, pp. 26-31.

Chang, H., Cooke, L., Hunt, M., Martin, G., McNelly, A., and Todd, L., 1999, Surviving SOC Revolution: A Guide To Platform-Based Design, Kluwer Academic Publishers.

Chung, M. K. and Kyung, C. M., 2004, Improvement of compiled instruction set simulator by increasing flexibility and reducing compile time, Proceedings of International Workshop on Rapid System Prototyping, pp. 38-44.

Chung, M. K., Yang, S., Lee, S. H. and Kyung, C. M., 2005, System-level HW/SW co-simulation framework for multiprocessor and multithread SoC, Proceedings of International Symposium on VLSI Design, Automation and Test, IEEE, pp. 177-180.

Clouard, A., 2002, Experiences and Challenges of Transaction-Level Modeling with SystemC 2.0, ST Microelectronics, presentation at the 5th European SystemC User Group Meeting.

Clouard, A. Jain, K., Ghensassia, F., Maillet-Contoz, L., Strassen, J.-P., 2003, SystemC: Methodology and Applications, Muller, W., Rosenstiel, W., and Ruf, J., ed., Kluwer Academic Publishers.

Cmelik, B. and Keppel, D., 1994, Shade: a fast instruction-set simulator for execution profiling, Proceedings of International Conference on Measurement and Modeling of Computer Systems, ACM SIGMETRICS, pp. 128-137.

Dynalith Systems, 2000, iSAVE User Manual, http://www.dynalith.com.

Dynalith Systems, 2002, iPROVE User Manual, http://www.dynalith.com.

Dynalith Systems, 2003, iPROVE SCE-MI Coemulation Manual, http://www.dynalith.com.

Dynalith Systems, 2004a, PhysicalModler, http://www.dynalith.com.

Dynalith Systems, 2004b, iPROVE AMBA Package Manual, http://www.dynalith.com.

Eiriksson, A.T., 1990, Mixed-level simulation with a Zycad simulation engine, ASIC Seminar and Exhibit, 1990. pp. P5/1.1-P5/1.5.

Gharsalli, F.; Meftali, S.; Rousseau, F.; Jerraya, A.A., 2002, Automatic generation of embedded memory wrapper for multiprocessor SoC, Design Automation Conference, pp. 596-601.

Grotker, T., Liao, S., Martin, G., and Swan, S., 2002, Chapter 8 Transaction-level modeling, in: System Design with SystemC. Kluwer Academic Publishers.

Keating, M. and Bricaud, P., 1999, Reuse Methodology Manual for System-On-a-Chip Designs, 2nd ed., Kluwer Academic Publishers, pp. 224-247.

Ki, A., Park, B.I., Lee, J.G., and Kyung, C.M., 2003, Cycle-accurate co-emulation with SystemC, SoC Design Conference, COEX ASEM Hall, Seoul Korea.

Ki, A. and Kim, Y.I., 2005, Reducing lock-step overhead of hardware-assisted simulation acceleration using protocol awareness, International SoC Conference, Seoul Korea.

Kim, N., Choi, H., Lee, S., Park, I.-C. And Kyung C.M., 1998, Virtual Chip:Making Functional Models Work on Real Target Systems, Design Automation Conference (DAC), pp.170-173.

Kim, Y.I., Yang, W., Kwon, Y.S., and Kyung, C.M., 2004, Communication-efficient hardware acceleration for fast functional simulation, Design Automation Conference (DAC), pp. 293-298.

Lee, J. Y. and Park, I. C., 2003, Timed compiled-code functional simulation of embedded software for performance analysis of SOC design, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 22(1):1-14

Nohl, A., Braun, G., Schliebusch, O., Leupers, R., Meyr, H. and Hoffmann A., 2002, A universal technique for fast and flexible instruction-set architecture simulation, Proceedings of Design Automation Conference, pp. 22-27.

OSCI, 2005, Draft Standard SystemC Language Reference Manual, http://www.systemc.org.

Pasricha, S., 2002, Transaction level modeling of SoC with SystemC 2.0, In Synopsys Users Group Conference India (SNUG'02), India.

Pees, S., Hoffmann, A., Zivojnovic, V. and Meyr, H., 1999, LISA-machine description language for cycle-accurate models of programmable DSP architectures, Proceedings of Design Automation Conference, pp. 933-938.

Rashinkar, P., Paterson, P., and Singh, L., 2001, 5.8 Simulation acceleration, in: System-on-a-chip Verification: Methodology and Techniques, 1st ed., Kluwer Academic Publishers, pp. 223-234.

Rosenstiel W., 2000, Chapter 3 Prototyping and emulation, in: Hardware/Software Co-Design: Principles and Practice, Staunstrup, J. and Wolf, W., ed., 1st ed., Kluwer Academic Publishers, pp. 75-78.

Schnarr, E. C., Hill, M. D. and Larus, J. R., 2001, Facile: a language and compiler for high-performance processor simulators, Proceedings of Programming Language Design and Implementation, ACM SIGPLAN, pp. 321-331.

Synopsys, 2003a, CoCentric System Studio User Guide, Version U-2003.03; http://www.synopsys.com.

Synopsys, 2003b, DesignWare AMBA SystemC Library User Guide; http://www.synopsys.com.

Synopsys, 2003c, DesignWare ARM SystemC Library User Guide; http://www.synopsys.com.

Wieferink, A., Kogel, T., Leupers, R., Ascheid, G., and Meyr, H., 2004, A system level processor/communication co-exploration methodology for multi-processor system-on-chip platforms. In Proc. Design, Automation and Test in Europe and Exhibition (DATE'04), Paris France, pp. 1256-1261.

Witchel, E. and Rosenblum, M., 1996, Embra: fast and flexible machine simulation, Proceedings of International Conference on Measurement and Modeling of Computer Systems, ACM SIGMETRICS, pp. 68-79.

Zhu, J. and Gajski, D. D., 2002, An ultra-fast instruction set simulator, IEEE Transactions on Very Large Scale Integration Systems, 10(3): 363-373.

Zivojnovic, V., Tjiang, S. and Meyr, H., 1995, Compiled simulation of programmable DSP architectures, Proceedings of Workshop on VLSI Signal Processing, IEEE, pp. 187-196.

Chapter 8

SoC TESTING AND DESIGN FOR TESTABILITY

Cheng-Wen Wu and Chih-Tsun Huang
National Tsing Hua University, Taiwan

Abstract: Integrating reusable cores from multiple sources is essential in system-on-chip (SOC) design. Testing these cores as well as the integrated system chip requires not just the conventional design-for-testability (DFT) methodologies, but also new ones. SOC testing involves applying test patterns to and analyzing the corresponding response from each and every core. In addition, the user-defined logic as well as the final integrated chip has to be tested. There are new challenges and issues, such as core isolation, test access, test pattern translation (from core to chip), test integration and scheduling, test automation, etc. This chapter discusses in detail the challenges and solutions in core-based SOC testing. We also briefly describe the IEEE 1500 that standardizes the test interface (called the *Test Wrapper*) between a core and its SOC host, and the *Core Test Language* (CTL) for test automation. We present a novel SOC test integration platform, solving real problems in test scheduling, test IO reduction, timing of functional test, scan IO sharing, embedded memory *built-in self-test* (BIST), etc. We also present a memory BIST compiler that provides a complete solution for SOCs with heterogeneous memory cores

Keywords: built-in self-test (BIST), Core Test Language (CTL), design-for-testability (DFT), IEEE 1500, memory testing, SOC testing, test access mechanism (TAM), test wrapper

1. INTRODUCTION

Testing is not a new engineering area, nor is it specific to electronic circuits. Not only man-made products (materials, devices, equipments, systems, etc.) but also natural goods need to be tested for their functionality and/or performance before we are confident of using them. The advent of semiconductor-based integrated circuit (IC) in 1958 created a new class of man-made products that require special techniques to test them. Over the years, electronic testing has evolved itself into one of the major electrical engineering fields, and produced industries in equipment, software tool, service, etc. Today, as we are entering the deep-submicron (DSM) age in the 21 century, system-on-chip (SOC) is becoming a reality. VLSI circuit chips are becoming so complex that their testing cost soars. Without continued research and development in test methodologies and technologies, test cost can rise to a level that becomes the bottleneck of developing and manufacturing new generations of VLSI circuits. In this chapter we will discuss the techniques and methodologies of SOC testing.

In addition to functionality and performance, the main purpose of VLSI testing is to guarantee the quality and reliability of the shipped parts of the circuit under test (CUT). Apparently, for that purpose we need to know what we are testing, i.e., we need to know 1) the types of defects and faults that can occur in a VLSI circuit, and 2) the types of circuits and circuit modules we are testing (digital, analog, memory, etc.). We then have to figure out how to test the defects and faults, and develop methods and tools to do that. This involves, in general, test pattern generation, test pattern application, and response evaluation. We also need to know the costs and effects of the methods and tools we develop. Specifically, the test quality and test cost have to be evaluated, i.e., we need to identify the relationship among the test cost, test coverage, and product quality and reliability. Developing cost-effective techniques, methodologies, and tools to guarantee product quality and reliability is the ultimate objectives of test development.

It is generally accepted that core-based and platform-based design methodologies are available for SOC design based on today's technology. As to SOC testing, test reuse and platform-based test methodologies still require investigation [1][2]. For an SOC, the design and test engineers may have to test the cores under a very limited knowledge of the core test information. The issues of core access and isolation are being addressed by the IEEE 1500 [3][4]. The IEEE 1450 [5] and IEEE P1450.6 [6] define the *standard test interface language* (STIL) and *core test language* (CTL), respectively, and provide a solution for test information exchange. Although the standards try to unify the core test wrappers and test information exchange format, the test controller, test architecture, *test access mechanism* (TAM), and test integration are left to the user—the SOC integrator.

Many TAM architectures (e.g., [7][8][9][10][11]) and test scheduling algorithms (e.g., [10][12][13][14][15]) have been proposed, however, there is little discussion that address test scheduling and TAM architecture at the same time. Most of the previous test scheduling works put emphasis on the reduction of test time without considering test architecture and TAM. Without considering practical test architecture, the test scheduling problem becomes unrealistic, where the cores can be accessed and tested at any time in any order. The scheduling result obtained that way is usually optimistic and impractical, requiring a complicated test controller, TAM bus arbiter, and/or massive test IOs. In our previous work [10], good scheduling result is obtained by using the session-based test scheduling approach and the *Test Access Control System* (TACS). However, the test time calculation in [10] is still too optimistic for real applications. The TestRail reported in [9] does not discuss the test controller and its complexity. Although the scheduling result is good, each TestRail requires its dedicated *Wrapper Serial Control* (WSC) signals. As a result, the number of test IOs can be high. Some other approaches such as the addressable test port (ATP) [8], CAS-BUS [7], and HD-BIST [11] provide flexible TAM and test architectures, but the drawback is high performance impact and area overhead.

In this chapter, we stress major issues in practical SOC test integration. First of all, test scheduling is defined by using a more precise model based on TACS. The realistic test time formulation reduces the complexity of test operations. With TACS, both the TAM bus arbitration and the control of test IOs can easily be done, and fewer test IOs are needed. The improved session-based test scheduling considers not only the realistic test control architecture and TAM bus, but also test IO limit. Issues on sharing and distribution of the test clocks and test enable signals are also discussed. Secondly, the coexistence of scan test and functional test is discussed in detail. The major challenge in applying functional patterns using the test wrapper is the timing requirement. Scan pattern application is relatively simple and straightforward by the 1500 parallel TAM, but at-speed functional pattern application is not as easy. We present a methodology to deliver the functional test patterns by using the scan architecture. In a legacy core the scan and functional IOs are usually shared, so there are timing problems in applying the tests through the 1500 Wrapper Boundary Register (WBR). With a minor modification of the WBR, we can solve the timing issues. The *Test Access Port* (TAP) Controller is extended and used as the Test Controller of the system chip, which is used to apply the scan tests, functional tests, or both to the embedded cores.

In addition to testing logic cores, we also discuss memory testing. Embedded memories are among the most common cores in system-on-chip (SOC) designs. The increasing demand for data bandwidth and the continuous decline in hardware cost makes embedded memory cores more and more popular for SOC applications. However, testing embedded memories is still a challenge since testing memory cores is much more difficult than testing commodity memories due to the limitation in available pins that can be used to access the cores, the increase in speed (due to the removal of off-chip loading), the increase in address and data bus widths, the availability of customized specifications and configurations of the memory cores, etc.

In addition to testing embedded memories using expensive external memory testers, built-in self-test (BIST) is considered a good alternative solution (see, e.g., [33][34][35][36][37][38][39]). With BIST, the overall test time can be minimized by parallel testing of the memory banks or blocks, and the external memory tester time can be greatly reduced. Since the test requirement is minimized, the cost of the tester reduces. However, a simple go/no-go BIST has limited applications because of the lack of diagnosis capability. Being able to provide the information such as the address and behavior of the faulty cells is an important feature that helps the user improve the memory design and process integration.

Embedded memories, unlike the commodity ones, are usually customized for different ASIC or SOC applications. The BIST circuits also need to be customized in such a case. An automatic BIST circuit generation tool will be required to increase productivity when embedded memory cores are frequently used. Several tools have been proposed for memory BIST circuit generation in the past. For example, a tool was proposed in [40] for automatic generation of both BIST and transparent BIST designs for memories. The methodology presented in [41] provides the automatic BIST design creation based on the popular march-based algorithms [35][42]. The BIST circuit detects port-coupling faults in multi-port RAM in addition to other common faults. A memory synthesis framework was proposed in [43][44], which can automatically generate, verify and insert programmable or non-programmable BIST circuitry in a short time. A memory BIST description language was used to help the integration of BIST and memory cores. Another example is our previous work—a simple programmable BIST compiler for EDO DRAM is reported in [45]. Almost all the proposed frameworks and existing commercial tools are designed for SRAM BIST only. We present a BIST compiler that supports both SRAM and DRAM.

Finally, we show at the end of the chapter an industrial SOC design that is developed using a test integration platform—*SOC Test Aid Console* (STEAC). The previous version of STEAC [16] has been greatly enhanced to support automation of test scheduling, test circuitry insertion and test pattern translation under the practical constraints as discussed above. Under the IO resource constraint, experimental results show that this approach is cost-effective. In addition, the cores with both scan and functional tests are supported by the enhanced TAP Controller and WBR.

2. IEEE 1500 AND TEST ACCESS CONTROL SYSTEM

Before discussing the test integration issues the IEEE 1500 Test Wrapper and Test Access Control System (TACS) [16] are briefly reviewed. We will show how to apply core test patterns, and present test time calculation under TACS.

2.1 IEEE 1500 [4]

To solve the problems mentioned above and for easy test automation, a standard test interface for the cores is required. A generic scalable architecture for SOC test is shown in Figure 1. [3], which was proposed by the IEEE 1500 Standard Working Group. The IEEE 1500 tries to standardize the Core Test Wrapper and the Core Test Language (CTL). The scalable architecture consists of

- the user-defined parallel *test access mechanism* (TAM) for delivering the test patterns and responses in parallel,
- standard core test wrappers that can isolate the cores and provide different test modes, and
- a user-defined test controller for controlling the wrapper and TAM [3].

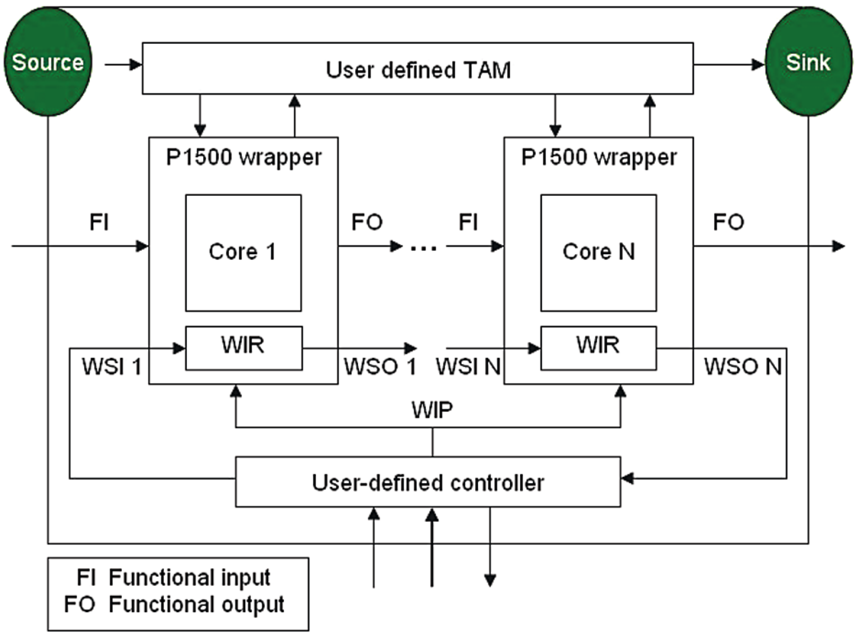


Figure 1. IEEE 1500 scalable architecture for SOC test

Off- or on-chip Source and Sink generate the test patterns and evaluate the test responses, respectively. Serial test access can always be done by using the *Serial Interface Layer* (SIL) provided by the 1500 Test Wrapper, which is mandatory. Note that adopting common test integration and optimization procedure is not necessary, and usually not possible, since the requirements and goals of the core providers, SOC integrators, and chip fabricators in the testing domain are different. Therefore, the TAM and test controller are user-defined. Among these components, the IEEE 1500 Standard Working Group only standardizes the test wrapper, and other components are designed by the SOC integrator.

Figure 2 gives the architecture of the IEEE 1500 Test Wrapper, which includes the following elements:

- *Wrapper Instruction Register* (WIR). This register decodes various test modes defined by mandatory and user instructions, and controls the operation of the WBR.
- *Wrapper Bypass Register* (WBY). Normally only 1-bit, this register directly connects the *Wrapper Serial Input* (WSI) to the *Wrapper Serial Output* (WSO). We use it to bypass the current core when we are testing other cores. If the core is not selected, we connect the output of WBY to WSO.

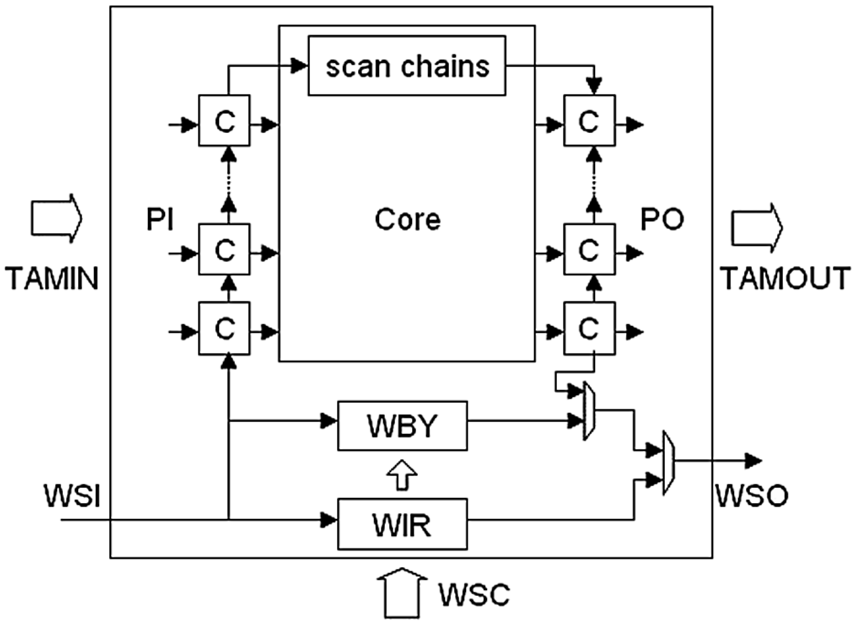


Figure 2. IEEE 1500 Test Wrapper architecture

- *Wrapper Boundary Register* (WBR). This register consists of the *Wrapper Boundary Cells* (WBCs) that wrap the cores' normal I/O pins, adding control, observation, and isolation capabilities to the cores' normal functions.

The Test Wrapper connects all functional inputs, scan chains, and functional outputs such that the test data can be shifted in through WSI and the test response can be shifted out from WSO. In general, the Test Wrapper has four major operation modes:

1. *normal mode*, in which the wrapper is transparent and the core operates normally;
2. *inward-facing mode*, in which the test access is for the core itself;
3. *outward-facing mode*, in which the test access is for the external circuitry; and
4. *safe mode*, in which the WBCs force the inputs of the core to a fixed pattern.

The first three modes are mandatory, and the last one is recommended. The IEEE 1500 also supports parallel test access to speed up the test process. In Figure 2, TAMIN and TAMOUT are parallel input and output ports, respectively. They are usually connected to a bus for parallel test data transfer. All inputs, outputs, and scan chains are connected to their assigned TAM bus lines, as shown in Figure 3. Note that TAMIN will connect the *primary inputs* (PI), scan chains, and *primary outputs* (PO) in the order shown. In this way, PI and PO can be overlapped to shorten the test time. In what follows, we assume the test data is transferred through TAMIN and TAMOUT, and the Test Wrapper is as shown in Figure 3.

At the chip level, we need to optimize the TAM and schedule the core tests [17]. Test wrapper and TAM co-optimization is important for the SOC integrator, since it has direct impact on the area overhead and ATE vector memory depth. The TAM is designed under the routing constraints between the cores and the system-level power constraints. The core tests are scheduled such that the total SOC test time can be minimized, subject to the testing power and area overhead constraints [18]. Test time reduction is done by exploring parallelism at both the chip and core levels [17].

2.2 Test Access Control System (TACS)

To speed up the development of system-on-chip (SOC), large and complex cores are being reused by designers. While design reuse is widely believed to effectively improve the productivity, test reuse is still a task that needs more effort than the designer can afford in general. The testing of SOC and the reusable cores (from different sources) has created some new

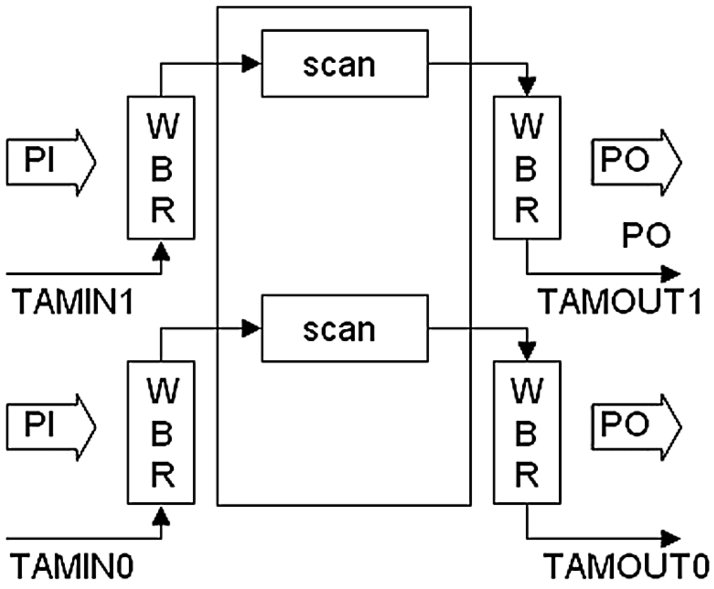


Figure 3. TAM routing example

challenges [1][2][21] for the designers and test engineers, such as 1) to test the cores with a very limited knowledge about the details of their test methodologies; 2) to access and isolate the deeply embedded core; and 3) to integrate and translate core tests to form the final SOC test. A typical SOC test design flow is shown in Figure 4. In the figure, we can see that the test issues include test information exchange, test wrapper generation, test access mechanism (TAM) design, test controller design, test scheduling, test integration, etc. The IEEE 1500 [3][4] defines the standard wrapper cells to support core test reuse and isolation. In [5][6], the *standard test interface language* (STIL) and *core test language* (CTL) are shown to provide the solution for test information exchange.

The TAM transports test patterns from the *test source* to the *core-under-test* (CUT), and transports the test responses from the CUT to the *test sink*. The TAM architecture affects the design of test wrapper and test controller, as well as the test scheduling algorithm. There were many TAM architectures proposed previously [8][22][23][24][25][26]. TAM architectures for cores with scan test were classified into multiplexing, daisy-chain,

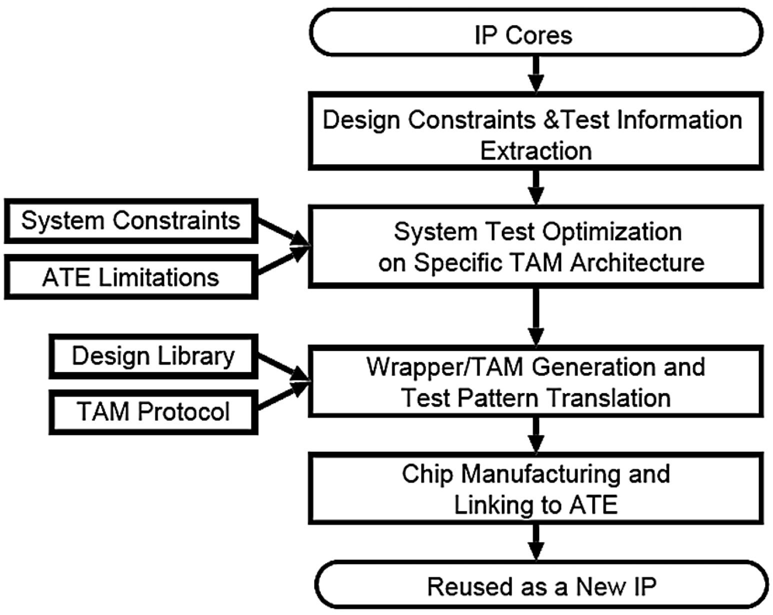

IP Cores
Design Constraints &Test Information Extraction
System Constraints
ATE Limitations
System Test Optimization on Specific TAM Architecture
Design Library
TAM Protocol
Wrapper/TAM Generation and Test Pattern Translation
Chip Manufacturing and Linking to ATE
Reused as a New IP

Figure 4. A typical SOC test design flow

and distribution architectures in [22]. In the *multiplexing architecture*, a multiplexer is added to the system to select the core to be connected to the TAM. In this architecture, only one core is tested at a time, resulting in long test time. In the *distribution architecture*, each core has one dedicated TAM, and can be tested at the same time without TAM resource limitation. In the *daisy-chain architecture*, the TAM forms a long scan chain over all cores. Bypass multiplexers are used to reduce the length of the shift path. With the bypass multiplexers, the cores can be tested in parallel, partly in parallel, or in series. It is more flexible than the distribution or multiplexing architecture. There are some other TAM architectures, e.g., a dedicated bus can be used for core test [23], and the on-chip system bus also can be reused as the TAM [24]. Also, in [25], the *TestRail* was proposed to combine the strength of both the test bus and boundary scan test. In TestRail, the multiplexing or daisy-chain architecture can be used. In addition, each core can have a dedicated TestRail, resulting in a distribution architecture. Other architectures such as *addressable test port* [8] and *CAS-BUS* [7] were proposed for high scalability and flexibility. They introduce higher hardware cost and performance impact. The choice of the TAM depends on the requirements of the SOC under test, including test time, performance penalty, area overhead, ease of test translation and test integration, etc. Some previous works have focused on the scheduling of core tests to reduce test time, subject to certain test resource and test power constraints [27][28][29][30]. The *test scheduling problem* has been reduced to some well known problems, e.g., the *integer linear programming* (ILP) problem, placement problem, and rectangle packing problem. However, so far little attention has been paid to on test integration at the system level.

In this sub-section, we present the *Test Access Control System* (TACS) that allows easy test integration for SOC. Based on the IEEE 1500 Test Wrapper, a TAM and the associated test controller are proposed, which are used to develop a test scheduling tool for optimizing the test time and/or TAM utilization. After test scheduling, the TACS hardware (including the core test wrappers, the TAM, and test controller) is automatically generated that meets the system test requirements. TACS also translate and integrate the core tests to the final system-level test automatically. It can be controlled via the IEEE 1149.1 *Test Access Port* (TAP) interface [19] or the IEEE 1500 *Wrapper Interface Port* (WIP). The TACS software has been developed based on the IEEE 1450 Standard Test Interface Language (STIL) and 1450.6 Core Test Language (CTL). The design specifications and test information of the cores and SOC are obtained from the user and the related CAD tools, while the test schedule is generated by our scheduling tool. The final SOC test is written in standard HDL and STIL for further use in physical design and ATE programming.

TACS contains a simple Test Controller that is compliant to the IEEE 1149.1 TAP Controller [32], and multiplexer-based TAM buses. The Test Controller will generate the WSC signals to operate the 1500 Test Wrapper for shift, capture, transfer, and update operations. The Test Controller also arbitrates the TAM bus for core test switching.

TACS is intended for managing three test tasks: 1) to control the core test wrapper operations, 2) to configure the TAM, and 3) to send the test patterns and receive the test responses. Although the Serial Interface Layer (SIL) defined in IEEE 1500 provides a standard test access mechanism for the cores, it is slow. A parallel TAM can be used to save the test time. In this section we will present the TACS hardware, including a hybrid TAM architecture and a test controller. Due to the similarity between SIL and IEEE 1149.1 TAP, reusing the TAP Controller as the SOC test controller is considered feasible [26][31]. However, the original TAP Controller does not handle hierarchical system test properly. The mixed use of TAPed cores and 1500 wrapped cores also complicates the test controller. The proposed TACS test controller has a unified interface for both the TAPed and 1500 wrapped cores.

Figure 5 shows an SOC architecture based on the proposed TACS, which consists of a system-level test controller (labeled TACS in the figure) and a

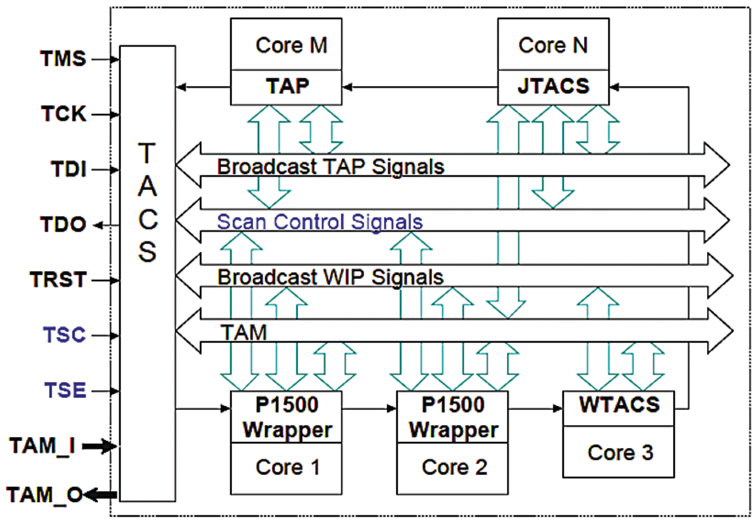


Figure 5. A TACS-based SOC architecture

parallel TAM (a test bus). To support hierarchical system test,all control signals of both the IEEE 1500 WIP and IEEE 1149.1 TAP are broadcast to the cores.

2.3 Test Pattern Application

Figure 6 gives the waveform regarding test data application in TACS. The figure describes how we handle the scan protocol by TAP states [32]. First, the TAP Controller enters the Pause-DR state by controlling TMS and enabling ShiftWR to operate the WBR cells in the shift mode (Load-Unload). After data are shifted completely, the TAP enters Exit2-DR and asserts UpdateWR so that the WBR cells update the core inputs (Force-PI). Then, the TAP goes to Capture-DR and asserts CaptureWR so that the WBR cells capture test response from core outputs (Measure-PO). After Measure-PO, the TAP enters Exit1-DR, and the scan clock is pulsed. Finally, the Controller returns to Pause-DR.

In Figure 6, TSE stands for *test scan enable* that controls all scan enable signals of the cores, and ScanClk and FuncClk are scan test clock and functional clock, respectively. Scan and input data are shifted in through TAMIN, while scan and output responses are shifted out through TAMOUT. Note that to fit the input timing of functional patterns, the Force-PI step takes 3 cycles to apply a functional input, and the functional clock (FuncClk) is applied during these three cycles.

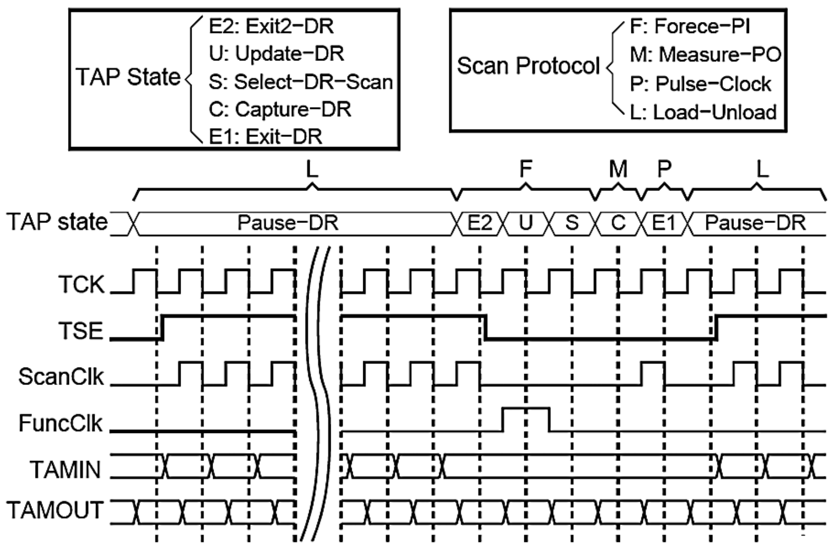


Figure 6. Test data application waveform

2.4 Test Time Calculation

If test time calculation does not reflect a realistic test application flow, the test time based on the scheduling result may not be really minimized. The test scheduler should calculate the test time based on the test application flow discussed above.

Test time calculation also depends on wrapper routing. Figure 3 gives a TAM routing example. Since loading the test input from PI and generating test response to PO can be done simultaneously, as shown in Figure 7, the shift length of the i^{th}-bit of TAM, L_i, is L_i=Max$\{PI_i,PO_i\}+S_i$, where PI_i, PO_i, and S_i are the numbers of PIs, POs, and scan flip-flops connected to the i^{th}-bit of the TAM, respectively. For a single core, the maximum L_i is the time to Load-Unload a test vector. After Load-Unload, it takes 5 cycles to capture output response (Force-PI, Measure-PO, and Pulse-Clock). If there is only one core in a test session, the test time of this session is $T=P \times (L+5) + L$, where P is the number of test vectors and L is the maximum shift length of TAM assigned to this core. In previous scheduling works [12][13][14][15], it is assumed that only 1 cycle is needed to capture the output response. For scan patterns, since the scan chains may be long, the number of cycles to capture output responses can be ignored. However, when functional patterns are considered, the number of available IOs is normally small, so the number of cycles to capture output responses is relatively large and will affect the scheduling results.

However, if there are multiple cores in a test session, test time calculation is different. Because all WBR cells are controlled by the global WSC signals, the cores with less test data must wait for other cores before the data are shifted completely. An example with two cores (A and B) in a test session is shown in Figure 7. Assume P_A (P_B) is the number of test vectors for Core A (Core B), and L_A (L_B) the number of cycles to transfer test data for Core A (Core B). Also, $L_A > L_B$. If $P_A > P_B$, the test time is the same as when only Core A is in this test session and there is no "Core B only" part in Figure 7. Each test vector needs L_A cycles to load and unload test data, and 5 cycles to apply Force-PI, Measure-PO, and Pulse-Clock. The test time is

$$T = P_A(L_A + 5) + L_A \tag{1}$$

If, on the other hand, $P_B > P_A$, the first P_A test vectors need L_A cycles to load and unload test data, while the remaining P_B-P_A test vectors need L_B cycles to load and unload test data. The test time is then

$$T = P_A(L_A + 5) + L_A + (P_B - P_A)(L_B + 5) \tag{2}$$

Based on the realistic test time calculation, we can schedule the core tests to really minimize the test time.

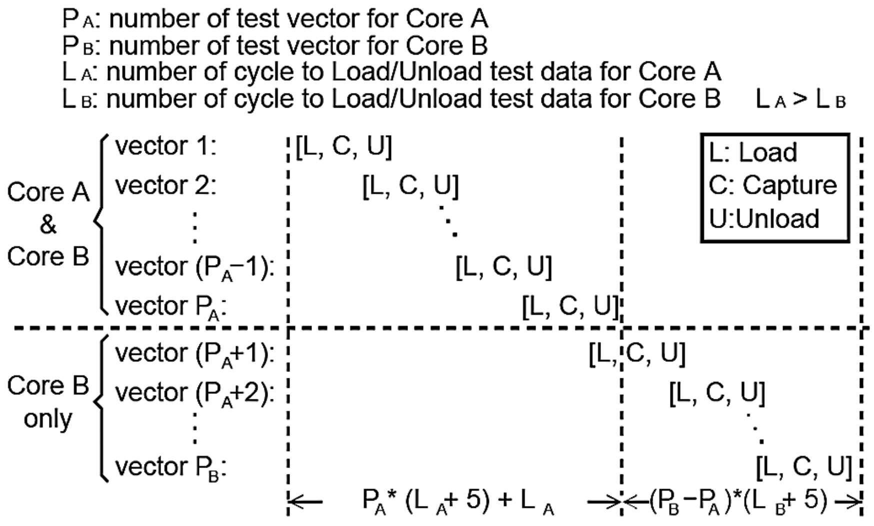

PA: number of test vector for Core A
PB: number of test vector for Core B
LA: number of cycle to Load/Unload test data for Core A
LB: number of cycle to Load/Unload test data for Core B LA > LB
Core A & Core B
vector 1:
vector 2:
vector (PA−1):
vector PA:
Core B only
vector (PA+1):
vector (PA+2):
vector PB:
[L, C, U]
L: Load
C: Capture
U:Unload
PA* (LA+ 5) + LA
(PB−PA)*(LB+ 5)

Figure 7. Test time calculation

3. TEST INTEGRATION ISSUES AND SOLUTIONS

Some test integration issues are discussed here. First, the *session-based test scheduling* should consider not only realistic test architectures and TAM bus arbitration, but also test IO resource constraints. As we have a limited number of test IO pins, it is important to share the test clock, test reset, and scan enable signals where possible. Reducing the test IO pins can increase utilization of the TAM bus for test data, leading to shorter test time. Another issue is how to apply functional patterns by the 1500 Test Wrapper, where timing is considered. The current 1500 core test environment is mainly designed for scan test. We will present an approach to delivering the functional tests by using the scan architecture in a more effective way, so far as timing is concerned. Finally, if some cores share scan and function IOs, there may exist timing problems when we apply the tests through the 1500 WBR. With a little modification of the WBR, we can solve the issue with minimum timing impact.

3.1 Scheduling Consideration

Most previous works on SOC test scheduling calculate the core test time as the product of the number of test vectors and shift-path length, assuming each of the cores can be tested at any time [12][13][14][15][10]. This assumption normally results in a shorter test time than the real case. The test time can be achieved only if each core has its dedicated WSC signals, which we normally cannot afford. On the other hand, if the WBRs share the WSC signals to reduce the test IOs, then the cores with shorter tests must wait for others before the test data can be shifted completely. Therefore, the test time is not really minimized. Another issue is that a complex test controller and TAM bus arbitration scheme would have been needed to switch the TAM bus among the cores in such an ideal case.

Our test scheduling method partitions cores into several *test sessions* to simplify the Test Controller design and TAM bus routing complexity. We use only a few test control IOs (i.e., TCK, TRST, TMS, TDI, and TDO) for test application. The WSC signals are broadcast to all cores, so all the WBRs shift, update, and capture data concurrently. The cores must wait for each other until all core test data are shifted completely, then they will do Force-PI, Measure-PO, and Pulse-Clock together. Only when all cores in one test session finish the test can we start another test session. Given the same TAM bus width, this approach results in longer test time than previously predicted [12][13][14][15][10]. However, the test IOs are also one key factor when considering the test cost. To implement the scheduling result as reported in previous works, many test IOs will be needed. With IO resource constraints, the test scheduling process is done in two steps: 1) a *coarse scheduling* is done to determine the number of test IOs and available TAM bus width; then 2) a *detailed scheduling* is done to optimize the test time. In general, the wider the TAM bus, the shorter the test time. Later we will compare the test time under the same IO resource constraint to justify TACS and the proposed session-based test scheduling approach.

3.2 Test IO Reduction

The clock, reset, and other test pins should be easily controllable for IP test, but the limited number of test IOs usually makes it a critical issue. For example, the p22810 benchmark of the ITC02 SOC benchmark suite [20] has 28 cores. Among these cores, there are 22 cores with scan chains. It means that at least 22 clock signals, 22 reset signals and 22 scan enable (SE) signals are needed to test these cores, if all cores are of single clock domain. In general, some cores have multiple clock, reset, and test signals. In that case, the number of chip IOs may be too small to accommodate the test IOs. In addition, test IO reduction will also reduce test cost, because the tester cost can be reduced, and under the same IO resource constraint, more IOs can be used for the TAM bus, so the overall test time can be reduced.

In TACS, all test pins (defined by TACS) of the cores are shared with functional IO pins (defined by designer). Dedicated test enable pins are not necessary—they can be generated by the Test Controller. We can share the test clock and reset signals for cores tested in different sessions. In Figure 8, the scheduling result allows 3 cores at most to be tested concurrently, assuming each core has only one clock domain. Only three test scan clocks (TSCs) are needed, i.e., core 1 (C1) and core 4 (C4) share TSC0, C2 and C5 share TSC1, and C3 has its own TSC3. In Session 0, TSC0, TSC1, and TSC2 are used to control C1, C2, and C3, respectively. While in Session 1, TSC0 and TSC1 are used to control C4 and C5, respectively. The test reset signals are similar to scan clocks. As the cores are isolated by the 1500 Test Wrappers, they will not be damaged due to shared test clocks or reset signals. By sharing the test clocks and reset signals, the number of test IOs is reduced.

To further reduce the number of test IOs, all scan enable (SE) signals are shared thanks to uniform test procedure in scan test. In general, the SE is enabled during the Load-Unload step and is disabled during Pulse-Clock (see Core 1 in Figure 9(a)). However, in some cases, the SE is still enabled even when the clock is pulsed (see Core 2 in Figure 9(a)). To share SE between both cores, we need to pulse the clocks of Core 1 and Core 2 in different cycles. The chip level test patterns are translated to pulse Clock1 in the Exit1-DR state and Clock2 in the Shift-DR state, just like the waveform

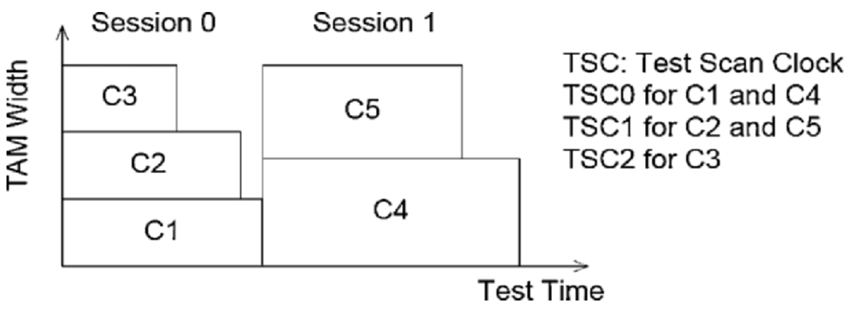


Figure 8. An example for test scan clock sharing

shown in Figure 9(b). This results in one-cycle test time overhead per scan vector. In general, hundreds or thousands of cycles are needed to shift in one scan vector, so the overall test time overhead is small.

Note that the translated test pattern is not exactly the same as the original test pattern. In the Force-PI and Measure-PO steps, the SE is disabled after translation, which is different from the original one. In most cases, the SE is simply used to switch the scan registers between the scan and normal modes. The slight difference of the test patterns does not affect the output response. If in a certain core the output response is sensitized by SE, then the core should have a dedicated SE.

Sharing SE signal may introduce another problem. In the Pulse-Clock step, if the SE is disabled, the scan cells will capture the data from the

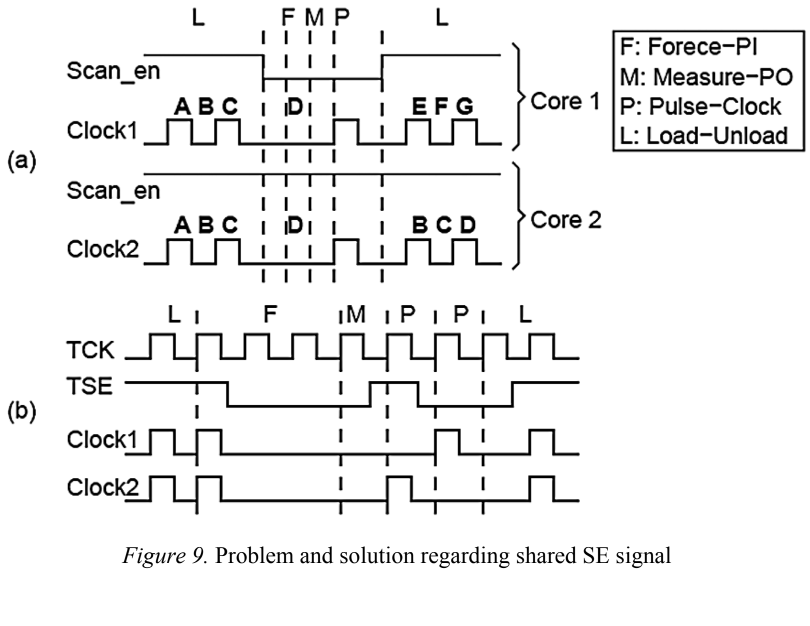


Figure 9. Problem and solution regarding shared SE signal

combinational block, otherwise each scan cell will capture the data from the previous scan cell (like shifting). For Core 1 and Core 2 in Figure 9(a), if the entered vector is ABC followed by D at the scan input, then after the Pulse-Clock step, the scan cells in Core 1 will store the data from the combinational part of Core 1, i.e., EFG. However, the scan cells in Core 2 will shift by one cycle, and the first scan cell will capture data D, so the data shifted out is BCD. When the TAM width assigned to this core is less than the number of scan chains, multiple scan chains are cascaded, and the scan data are sent through a one-bit TAM. To apply data D to the cascaded scan chains, a flip-flop is inserted in between any two cascaded scan chains. The vector shifted in becomes ABCD, while data D is stored in the additional flip-flop and is shifted into the scan chain after the Pulse-Clock step.

3.3 Timing Issues in Functional Test

During core internal testing, the WIR will generate shifting, updating and capturing control signals based on the WSC signals to all WBR cells concurrently, so the WBR cells will shift, update, and capture test data in synchrony. However, all inputs of the *functional patterns* do not always have the same timing waveform. Consider the inputs IN_A, IN_B and IN_C as shown in Figure 10. IN_A is available before the rising clock transition; IN_B is available after the rising clock transition; and IN_C is available after the falling clock transition. To satisfy the timing relationship, a delay

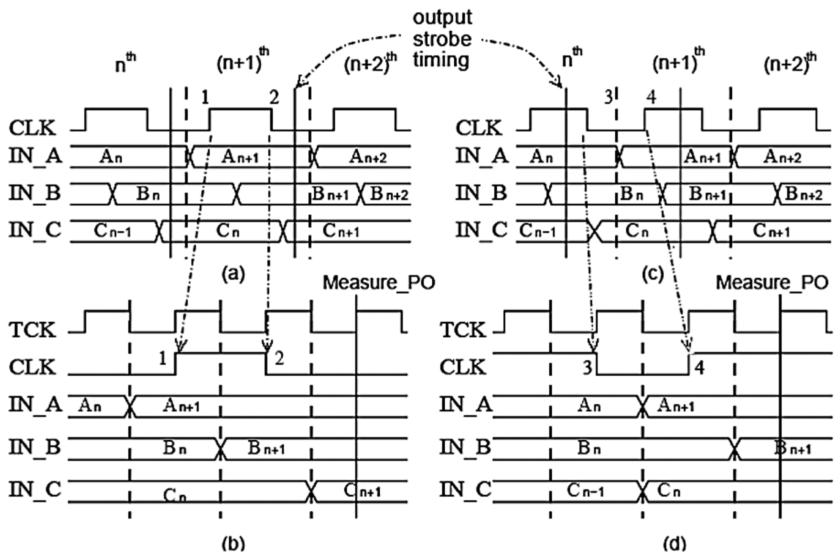


Figure 10. The timing of cycle-based functional patterns

element is added to the updating control signal connected to the WBR cells of IN_B and IN_C. Figure 11 shows the modification, where i_u is the updating control signal for the input wrapper cells generated by WIR. With the delay element, the test data of IN_B and IN_C will be updated with one- and two-cycle delays, respectively. The system clock rises between the updating times of IN_A and IN_B, and falls between the updating times of IN_B and IN_C. Figure 10(b) shows the waveform of the transformed test vectors. With the additional delay elements, the input data are available in the corresponding order. If the input data are available after the output is strobed (see, e.g., IN_C in Figure 10(c)), the data of the previous vector are used. The *SOC Test Aid Console* (*STEAC*) [16][18] is a test integration tool that supports the flow shown in Figure 4. It parses the functional patterns to get the input timing information, and connects the updating control signals with corresponding delay to each input WBR cell to fit the input timing.

Note that it is impossible to update the test input data at exactly the same time as the original waveform, because the WBR control signal generation is aligned with the clock cycles. Therefore, we just present a solution to applying the functional patterns in the *scan-based test* environment. Note also that the test clock is not applied at system speed, as the scan patterns of other cores in the same test session are applied at a lower speed. More effort is needed to verify the timing of *functional test*.

Another issue of functional patterns is the clock signal in the first vector. In Figure 12, before output strobe, the clock signal rises in the first vector, and toggles in the following vectors. Test pattern translation must also check

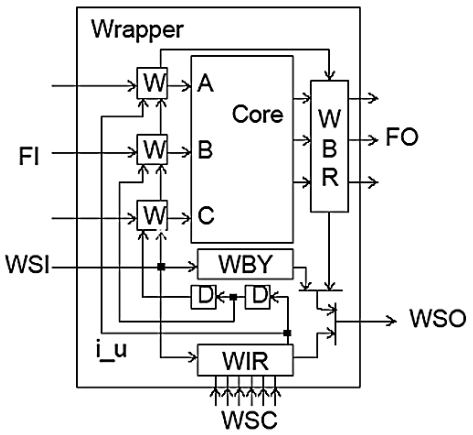


Figure 11. Test Wrapper supporting functional patterns

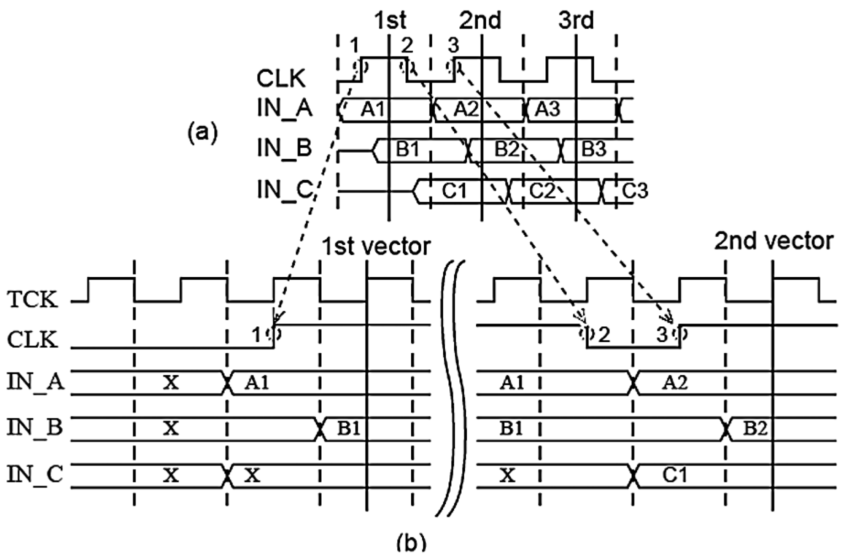


Figure 12. First two vectors of the functional pattern

the initial state of the clock signal to keep the translated waveform the same as the original functional waveform. In this way, TACS supports any functional patterns with different input timing.

3.4 Scan and Functional IO Sharing

In most legacy cores, the scan chains and functional IOs may share the core IO pins. This is for reducing the number of original chip IO pins. However, when a legacy core is integrated into the SOC, IO sharing of the legacy core becomes a problem during test pattern application. Figure 13 gives a few examples for IO sharing between scan chains and functional IOs. In the figure, we assume scan1 and scan2 are cascaded by TAMIN1 and TAMOUT1, while TAMIN0 and TAMOUT0 connect all WBR cells of the PIs and POs. Figure 13(a) shows the solution for output sharing between the scan chains and functional IOs. In this example, scan1 and scan2 share the outputs PO1 and PO2 with the functional IOs. The PO2 terminal is wrapped by the WBR cell, WBRC4, and is also connected to the scan-in terminal of scan2. The PO1 terminal is wrapped by WBR3 and is also connected to TAM-out. In the Measure-PO step, WBRC3 and WBRC4 capture the functional outputs of PO1 and PO2. In the Load-Unload step, the scan data of scan1 are shifted out to scan2 through PO1, while the scan data of scan2 are shifted out to TAM output through PO2. These shared outputs are used as either the functional data outputs or scan data outputs in different steps.

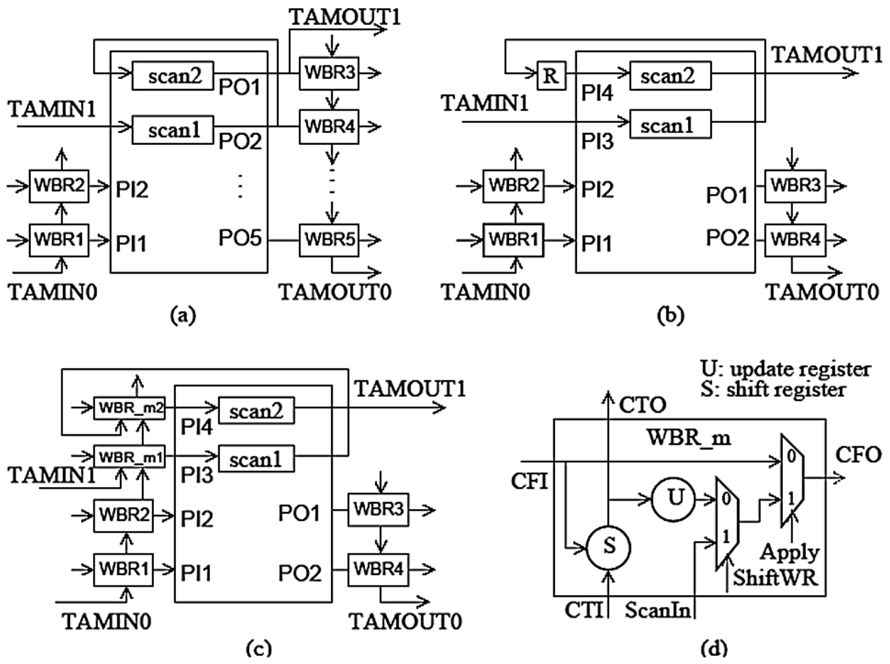


Figure 13. IO sharing: (a) output sharing, (b) input sharing with only scan tests, (c) input sharing with both scan and functional tests, and (d) modified WBR cell

The solutions for scan/function input sharing are dependent on the type of test patterns. If there are only scan patterns, a flip-flip is added to the shared pin, in addition to the WBR cell. Figure 13(b) shows an example, where scan1 and scan2 share the inputs PI3 and PI4 with the functional IOs. In the Force-PI step, the data of PI3 input can be applied by TAMIN1, but the data of PI4 input can not. To apply the PI4 data, a register cell (the R flip-flop as shown in Figure 13(b)) is added in between scan1 and scan2. The input data of PI4 is shifted in with the scan data and is applied by the added flip-flop. If there are scan and functional patterns, the solution is shown in Figure 13(c) (the routing example) and Figure 13(d) (the modified WBR cell). PI3 and PI4 are wrapped with the modified WBR cells, labeled as WBR_m1 and WBR_m2. The additional path from ScanIN to CFO allows the scan data to be shifted in through this type of WBR cell. The additional multiplexer is controlled by the ShiftWR signal. During the Load-Unload step, the ShiftWR signal is enabled and the functional IO data can be shifted in through the CTI-CTO path, while the scan data can be shifted in through the ScanIn-CFO path. In the Force-PI step, the ShiftWR signal is disabled and the functional data is updated through the Update flip-flop to CFO. With the modified WBR cell, the test vectors can be shifted in correctly even when the scan chains share inputs with the functional IOs, and the delay between CFI and CFO is reduced to a multiplexer delay.

The second solution also can be used for a core with only scan patterns. In this case, the first solution will make the scan length one-bit longer, while the second solution will increase the area overhead a little (by using the modified WBR cell).

3.5 STEAC: SOC Test Aid Console

Figure 14 shows the SOC test integration system called STEAC—*S*OC *Te*st *A*id *C*onsole [16], which consists of four modules: the STIL Parser, Core Test Scheduler, Test Insertion Tool, and Pattern Translator. It solves the test integration issues discussed above.

The *STIL Parser* parses the test information of each IP. The test information is written in STIL and is generated by commercial ATPG tools. Therefore, STEAC can be integrated into a typical design flow easily. The test information includes IO ports, scan structure (number of scan chains,

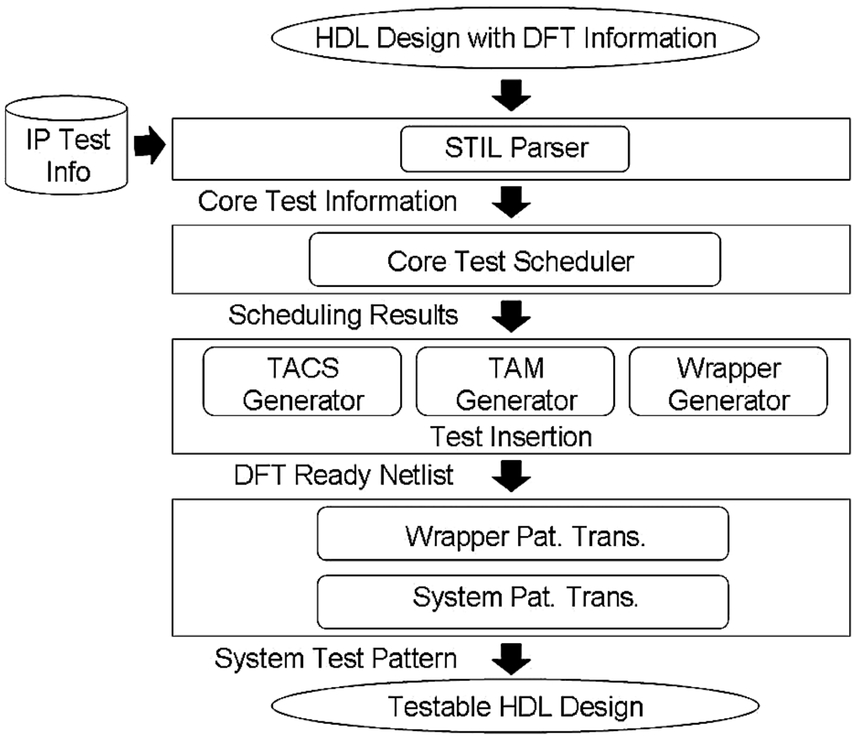


Figure 14. Test integration flow of STEAC [16]

length of each scan chain, etc.), and test vectors. With the core test information, *Core Test Scheduler* will schedule the core tests to reduce the overall test time. The Scheduler partitions core tests into several test sessions, and assigns the TAM wires to each core to meet the power and IO resource constraints. If the IP is a soft core, the scan chains can be reconfigured. The Core Test Scheduler will then rebalance scan chains for each assigned TAM width. The results can be fed back to the SOC integrator to reconfigure the scan chains to balance the chain length. The scheduling results are also used to generate the Test Controller, TAM bus, and Test Wrapper. Finally, the generated test circuitry is inserted into the original SOC netlist automatically. A new SOC design with DFT will be ready in minutes.

The core test patterns are generated at the core level. After the cores are wrapped, the test patterns must be translated to the wrapper level and then to the chip level. The test patterns are cycle based, which can be applied by external ATE easily.

3.6 BRAINS

As embedded memories are handled in a different way from that for logic cores, we also present here a memory BIST compiler called *BRAINS* (*B*IST for *RA*M *in S*econds), which supports SRAM and DRAM by using a novel BIST template approach. It generates the BIST design in synthesizable Verilog HDL upon receiving the memory specifications and test requirements provided by the user. The synthesizable BIST core can then be optimized for different fabrication processes. BRAINS also generates scripts for a commercial synthesis tool that performs timing validation for the BIST circuit during the synthesis process. The BIST design provides *at-speed testing* and *diagnosis* support, and is programmable for various *march tests*. BRAINS thus can be used for generating BIST circuits that target different RAM-core architectures and configurations.

In addition, BRAINS supports automatic test integration of multiple and heterogeneous memory cores in an SOC environment. The BIST architecture is improved for parallel testing, multi-core diagnosis, and *on-chip bus* (*OCB*) interface [48]. The proposed *test grouping and scheduling* (*TGS*) algorithm facilitates BIST generation under various test constraints, such as test time, test power, and other user-defined constraints. Finally, BRAINS is equipped with a *graphical user interface* (GUI), and the BIST generator can be integrated with a memory compiler to form an IP (intellectual property) generator for various memory configurations.

3.6.1 BRAINS Templates

BRAINS generates BIST circuits by using various *BIST templates* that provide building blocks for march-based testing. In practice, design migration cannot be done by simply changing the parameters. It usually requires detailed adjustment, especially for DRAM cores. The allowed adjustment space for the *BIST compiler* has to be defined within the BIST templates. Three different templates are defined: 1) the *Controller*, 2) the *Sequencer*, and 3) the *Test Pattern Generator* (TPG). We use the templates to construct the BIST architecture for the embedded memories, as shown in Figure 15.

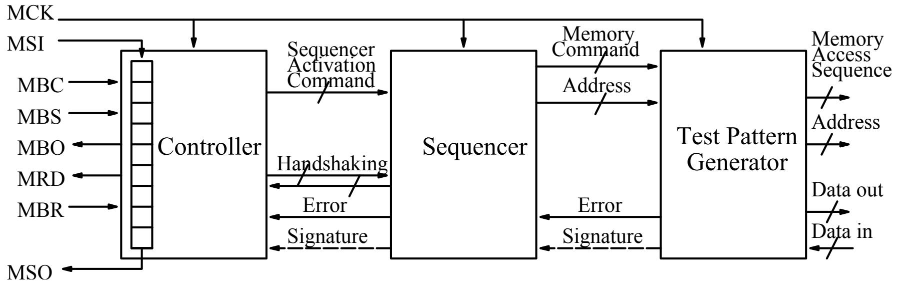


Figure 15. BIST architecture using the templates

During a *march test*, the Sequencer generates the address sequence (either ascending $\Uparrow$ or descending $\Downarrow$ [35]) and various *memory access commands* based on the specifications of the memory under test. For example, the Sequencer may generate the *read*, *write*, *refresh*, *precharge*, *load_mode_register*, *active*, and *nop* (no operation) commands for an SDRAM, and the *read*, *write*, and *nop* commands for a single-port SRAM. Some standard memory access commands for typical memory types are defined in the memory library, but customized commands specified by the user can be included as well. The Sequencer generates high-level commands rather than the low-level (physical) access commands.

The Sequencer architecture is shown in Figure 16. The Control Module receives march commands from the Controller. It controls the Address Generator, Sequence Generator, and Memory Command Generator. The Address Generator generates ascending and descending address sequences as specified by the march elements. The Sequence Generator generates the access sequence in the march elements. The optional Error Handling Modules in the Sequencer and the TPG are used to scan out the error address, error signature, and the corresponding march operation that activated the fault to the external tester for diagnosis and analysis.

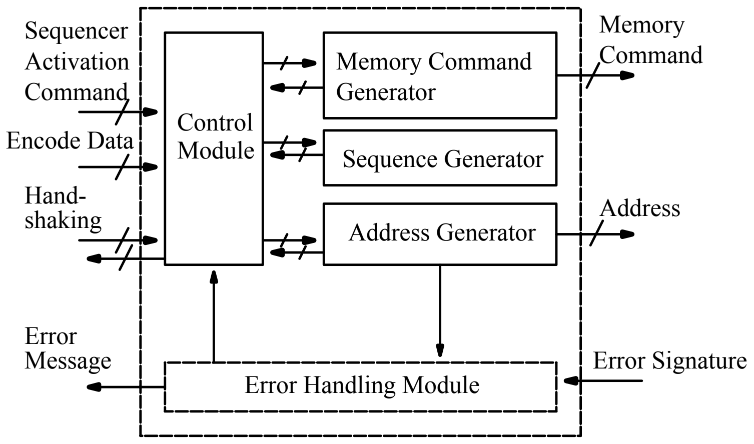


Figure 16. Block diagram of the Sequencer

The TPG converts high-level memory access commands from the Sequencer to low-level (physical) timing, address, and data sequences that can be sent directly to the memory core. The timing, address, and data sequences can be high-speed, double-edge triggered, packetized, or even of different signal levels. The TPG also compares the data output (Q) from the memory with the original data pattern (D) to determine whether an error exists. In the diagnosis mode, the Error Handling Modules are used to scan the error signature out. Both the Sequencer and the Test Pattern Generator are highly modularized.

3.6.2 Configuring the BIST Templates

BRAINS can generate the BIST circuit according to configurations specified by the user, such as fast access mode and diagnosis support. For example, to maximize the data bus utilization, an *interleaved access mode* in SDRAM BIST can be specified. The timing sequence (waveform) of a read-write *march element* (e.g., $\Uparrow(raw\overline{a})$) is shown in Figure 17 for a four-bank SDRAM which has shared D and Q bus and a *CAS latency* of 3. Three nop operations are required between the read operation and the write operation if the march element is performed linearly as shown in Figure 17. In the figure, the column address changes from *i* to *i*+3. The user can specify interleaved bank access as shown in Figure 18, which reduces the test time. In the figure, four consecutive read operations are performed before the four consecutive write operations by interleaving the bank addresses, so only three nop operations are required in between.

Table 1 lists the clock cycles required for accessing four addresses. We compares the non-interleaved and the interleaved cases for two different

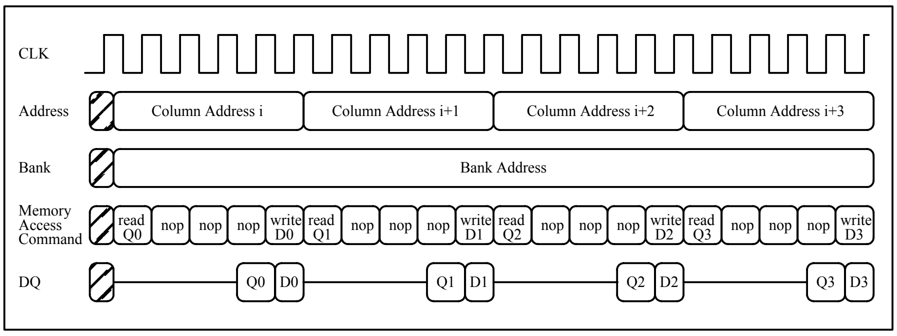


Figure 17. Non-interleaved bank access for the SDRAM example

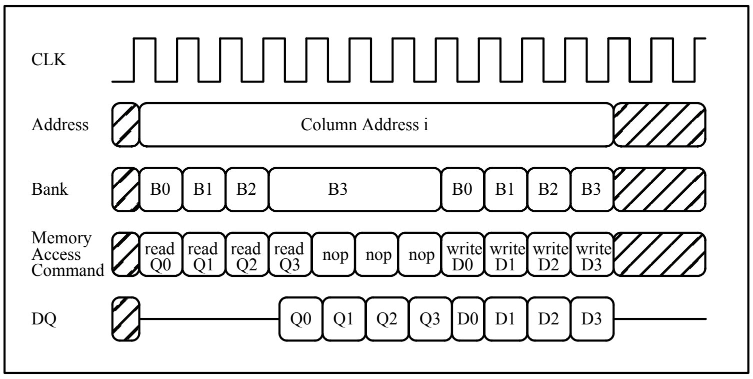


Figure 18. Interleaved bank access for the SDRAM example

march elements (i.e., $raw\bar{a}$ and $raw\bar{a}r\bar{a}$, where a can be 0 or 1) and two different CAS latencies (i.e., 3 and 2). In the case of separate D and Q buses, however, the non-interleaved bank access should be used since it results in a smaller area overhead with the same test time. The BIST templates can be configured under different timing constraints to minimize the hardware and test costs. This is very useful for customized memory cores.

Another option in BRIANS is the *diagnosis support* (for circuit debugging and/or repair analysis). If diagnosis is specified, the Error Handling Module

Table 1. Clock cycle comparison of four-address read-write operations for a four-bank SDRAM

	CAS Latency = 3			CAS Latency = 2		
March Element	Non-Interleaved	Interleaved	Reduction	Non-Interleaved	Interleaved	Reduction
$raw\bar{a}$	20	11	45%	16	10	37.5%
$raw\bar{a}r\bar{a}$	24	15	37.5%	20	14	30%

will be inserted. The user will then be able to switch between the *BIST mode* and the *diagnosis mode* by using different test commands. In the BIST mode, the BIST circuit accesses the memory in a pipelined way and only reports the go/no-go result (i.e., whether the memory functions correctly or not). In the diagnosis mode, if an error occurs, the Error Handling Module will scan out the error address, error signature and the corresponding march operation through the MSO pin, when MBO is pulled down that indicates the scan out operation (see Figure 15 and Figure 16).

3.6.3 BIST Architecture for Multiple Memory Cores

Figure 19 shows the BIST architecture for multiple memory cores. The external tester can access all the memories via a single shared BIST controller. One or more Sequencers can be used to generate march-based test algorithms [35][50]. Each TPG attached to the memory will translate the march-based test commands to the respective RAM signals.

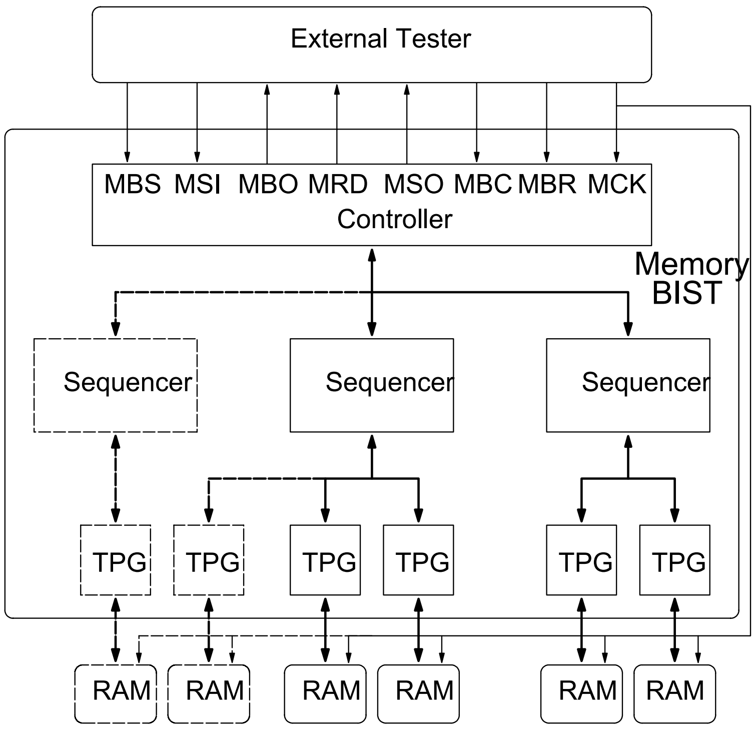


Figure 19. BIST architecture for multiple memory cores

Memory cores with similar timing specifications can share the same Sequencer to reduce hardware overhead. Moreover, memories close to each other can be grouped, subject to user-specified constraints. The TPGs can test the corresponding memories concurrently

The Controller is shown in Figure 20, which consists of a Control State Machine and a Command Storage Chain. The Control State Machine has four states: Idle, Select, Scan, and Run. A unified BIST interface makes future extension possible, reducing the control complexity of the test host [39][47]. The Command Storage Chain is basically a serial-to-parallel FIFO. For the purpose of SOC memory core testing, the command storage consists of six fields: 1) Mode—to select the operation mode, such as parallel test, individual test, diagnosis, or repair support; 2) Sequencer ID—to activate the target Sequencer; 3) Group ID—to activate the target group in the Sequencer; 4) Member ID—used when testing an individual memory; 5) Data Background—an encoded pattern for the march test; and 6) March Commands—encoded march elements as presented in [39][47]

The Controller receives test commands and generates error signatures serially to reduce IO overhead. However, the serial data requires additional serial-to-parallel and parallel-to-serial converters. For SOC designs, it is more feasible to access the BIST by an embedded test host, such as the processor. With an OCB wrapper, the BIST can be attached to the existing OCB. During the diagnostic process, the parallel bus-based interface simplifies the control complexity and saves the test time dramatically.

We present an OCB interface for the popular *Advanced Microprocessor Bus Architecture* (*AMBA*) as an example. Note that AMBA is an open standard [49]. Figure 21 shows the AMBA-based interface for an *Advanced Peripheral Bus* (*APB*, part of AMBA) wrapper. The APB wrapper

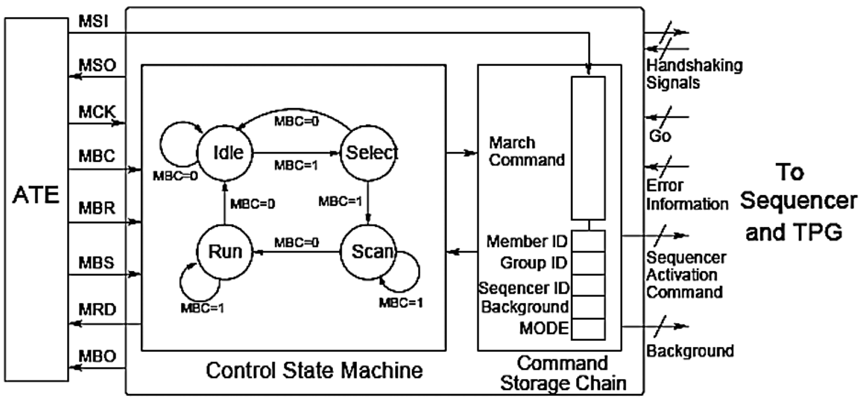


Figure 20. Block diagram of the Controller

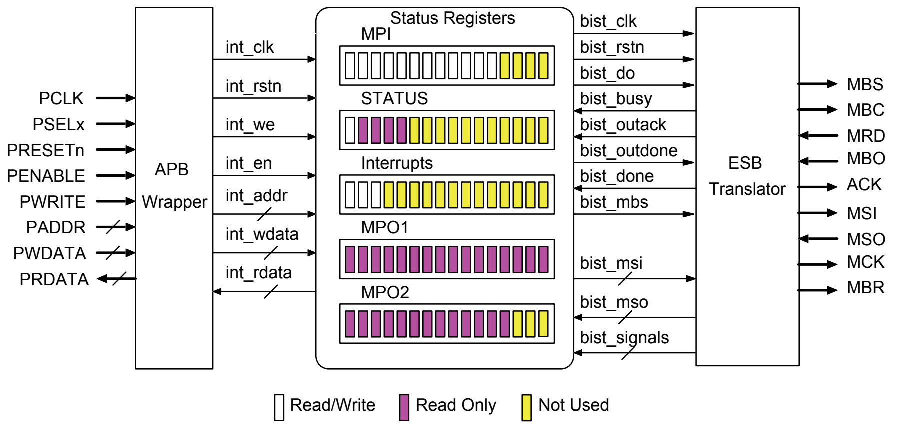


Figure 21. An OCB interface example: the APB wrapper

implements the standard three-state protocol to transfer commands and data between the test host and the BIST Controller. The Status Registers block stores the test data such as test commands and error signatures. Four different status registers are defined: 1) MPI—to load the test commands in parallel; 2) STATUS—to indicate the Go/No-Go, handshaking, and BIST internal status; 3) Interrupt—to keep track of the interrupts; and 4) MPO1 and MPO2— to store the error signatures. The width of the APB data is 16 bits, so are the status registers, but they can be extended easily. For example, in Figure 21, there are two MPO registers for the 29-bit error signature using the 16-bit APB bus. The number of registers varies according to the width of test commands and error signatures. The overhead is very small since most of the registers share the flip-flops with the signals from the Controller

The Sequencer (see Figure 22) receives test and control commands from the Controller and generates march sequences for the TPG. Additionally, the TPG Selection Module is for selecting the proper TPGs and RAMs. It enables and disables the proper memories according to the address space and cycle time for the respective march tests. The OR Module merges the Go/No-Go signals from the TPGs. When in diagnostic mode, the Sequencer has to capture the error signatures from the defective memory through the optional Error Handling Module.

The TPG receives march sequences from the Sequencer and accesses the associated memory core directly. Figure 23 illustrates the configurable architecture of the TPG. The Command Converter translates the march sequences into RAM signals, while the Command Buffer adjusts the timing delays of the outputs. The Port Selection Module enables and disables the port under test. The outputs from the memory are also buffered before entering the TPG. The additional command and data buffers allow the BIST to run at speed in a pipelined fashion.

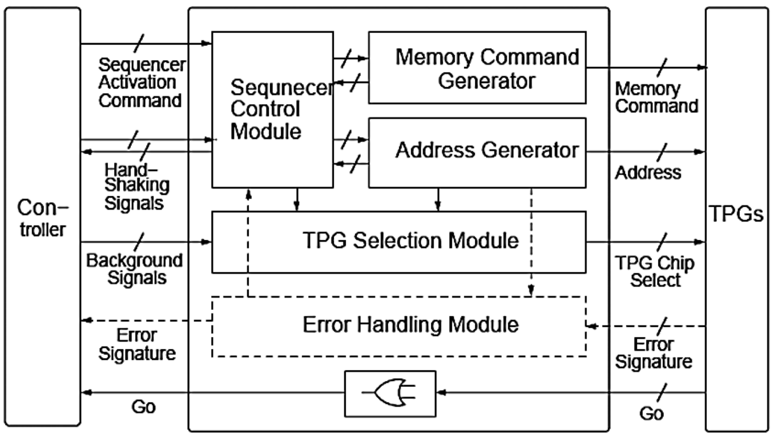


Figure 22. Block diagram of the Sequencer

The Comparator detects the faulty outputs and produces the *Go/No-Go* signal. The optional Error Handling Module captures the error signature for diagnosis. The error information is scanned out to reduce routing overhead. The target memory cores are accessed one by one in the diagnosis mode. Finally, the optional *Test Collar* allows the memory to switch between normal and test modes. Note that the collar usually is integrated with the memory core to reduce performance penalty.

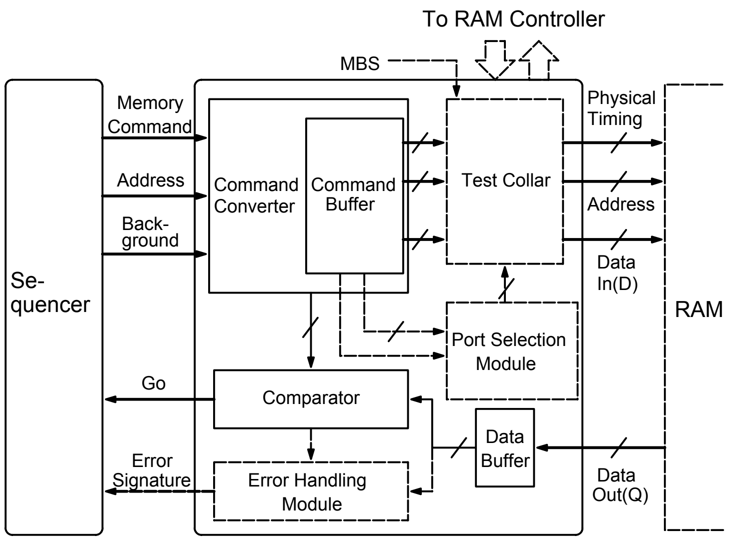


Figure 23. Block diagram of the TPG

The TPG needs slight modification to handle *dual-port SRAM*, *multi-port SRAM*, and *n*-read-*m*-write *register files*. Figure 24 shows the connection configurations between the Command Buffer and the memory for three different memory types. For a dual-port memory, the shared TPG performs the test one port at a time. Testing the multiple-port memories is similar. For an *n*-read-*m*-write register file, a pair of read port and write port forms aread-write port during the testing process. Note that a shared TPG architecture has a lower area overhead, but it does not detect complex inter-port faults.

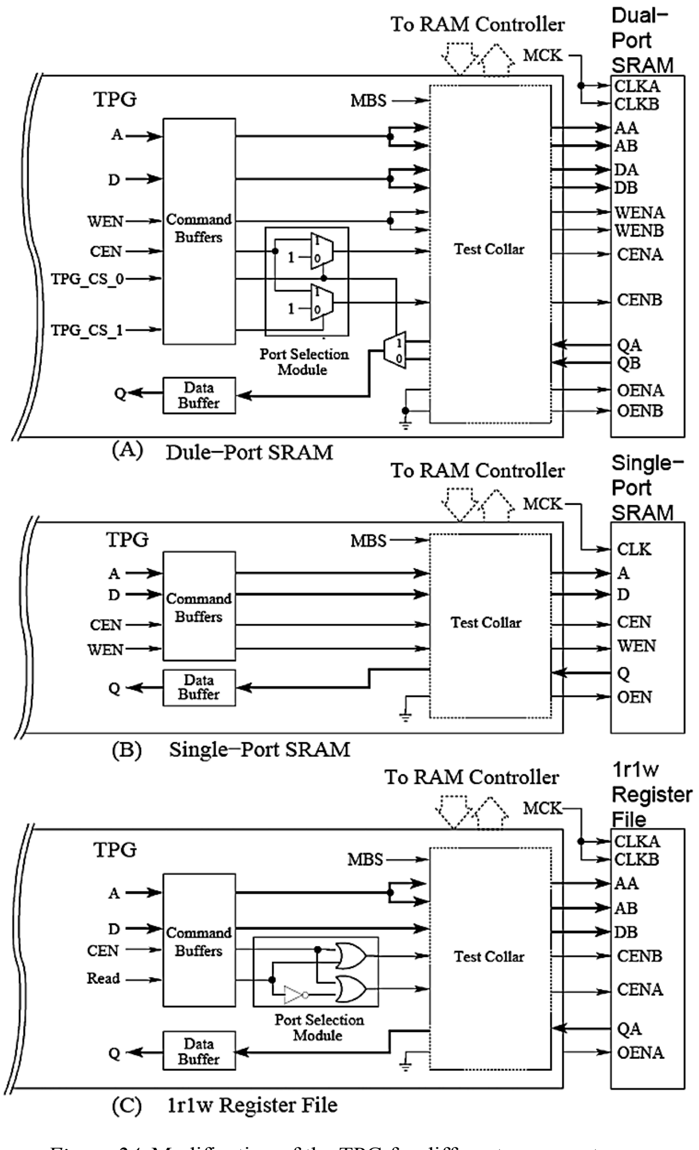


Figure 24. Modification of the TPG for different memory types

3.6.4 Test Grouping and Scheduling

The purpose of test grouping and scheduling is to minimize the overall testing time for all the memory cores, given limited test resources. The *test grouping and scheduling* (*TGS*) algorithm requires slight modification of the BIST design. It has simple control and negligible hardware overhead.

For each memory we define four attributes: 1) the size of the address space S_a (i.e., the maximum address); 2) the number of *data background* words n_B for testing the word-oriented memory (normally $\lceil \log w \rceil$ for a w-bit word); 3) the port index n_P (i.e., the number of read-write ports); and 4) the schedule step E_s (i.e., $2^{\lceil \log S_a \rceil}$). For each test group, we denote the maximum S_a among the memories in the group as S_m, and the size of the schedule space as $S_s = 2^{\lceil \log S_m \rceil}$. In the schedule space, each memory can be scheduled only by its step size E_s, and the number of step points is $\lfloor (S_m / E_s) \rfloor$. The TGS algorithm is shown as follows (see also Figure 25).

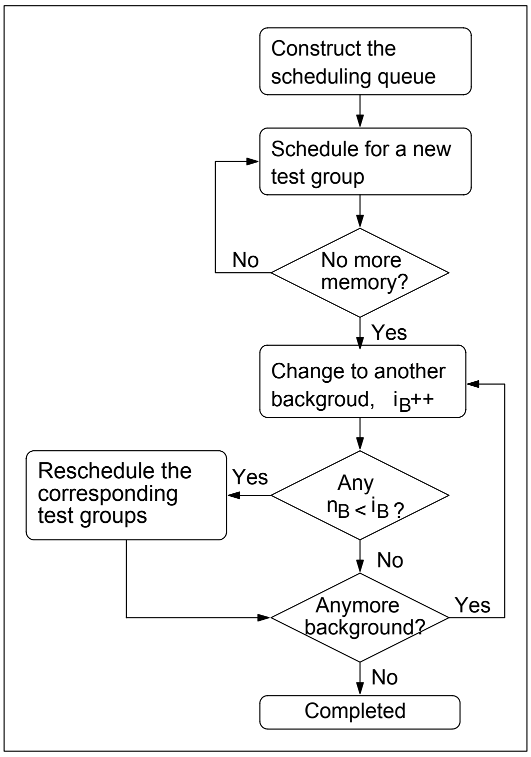

Construct the scheduling queue
Schedule for a new test group
No more memory?
No
Yes
Change to another backgroud, iB++
Any nB < iB ?
Yes
Reschedule the corresponding test groups
No
Anymore background?
Yes
No
Completed

Figure 25. Flowchart for the TGS algorithm

1. Select the solid (all-0) background, and let the background index $i_B = 0$. Let Π be the pool of all *k* unscheduled memories M_j, $0 \le j \le k-1$.
2. Schedule a new test group:
   (a) Get a memory with the largest S_a from Π. Remove the memory from Π. Let $S_m = S_a$ and calculate S_s for the test group.
   (b) Get a memory from Π according to the priority of S_a and n_P (i.e., high S_a to low S_a for multi-port memories first, and then high S_a to low S_a for single-port ones). Search among the schedule space. For a single-port memory, start the search from the origin of the schedule space, while for a multi-port one, start the search from the end of the last scheduled memory test. This is done under some user-defined constraints such as power and area overhead. If the schedule is unsuccessful, put the memory back to Π and select the next one.
   (c) For a successful schedule, $n_P = n_P - 1$. If $n_P = 0$, remove the memory from Π.

   Repeat (a)-(c) until no more memory can be scheduled for the current test group. If all the unscheduled memories can be tested concurrently under the given constraints, scheduling is neglected for the test group to reduce the control overhead.
3. Repeat Step 2 until all memories are grouped.
4. Schedule the test for another data background, and $i_B = i_B + 1$. Let the test groups be the same as for the previous background.
5. If there are memories in a group such that $n_B < i_B$, remove all such memories from that group. For each test group whose S_m decreases, apply Step 2(b) to reschedule the group, and if the reschedule is unsuccessful, increase its S_s by the smallest E_s in the test group and repeat Step 2(b) again.
6. Repeat Steps 4 and 5 for all data backgrounds.

Figure 26 shows an example of the TGS algorithm for a five-memory case. Assume the power weights of memories A to E are 200, 150, 140, 130, and 100 units, respectively, and the power limit (constraint) is 400 unit. For the solid background, the S_m and S_s of Group 0 are both 8 (from memory A), as shown in Figure 26(a). The dual-port memory B is then scheduled at time 0 (i.e., the address counter has a value 0), and its port index n_P is decremented. Memory C is scheduled at time 4 after the schedule space search, under the power constraint. Memories B, D, and E are then put into Group 1 without scheduling, since their total power consumption is only

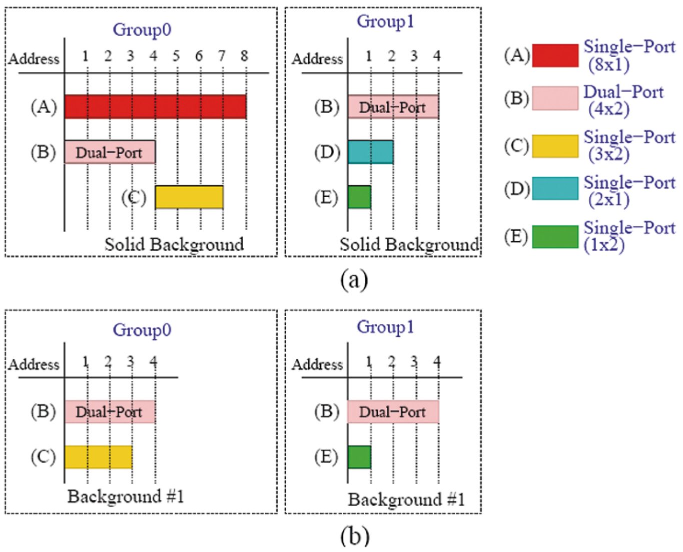


Figure 26. An automatic scheduling example

380units. In Group 1 the memories are accessed in parallel, so the control circuit is simplified. For the second background, testing the bit-oriented memories A and D is unnecessary. After removing memories A and D, Group 0 is rescheduled to reduce the test time, while Group 1 remains the same. The Sequencer will generate the control signals to enable or disable the memory cores based on the scheduling result. Consequently, the proposed TGS algorithm provides a systematic way to grouping and scheduling the entire test process efficiently.

3.6.5 BIST Circuit Compilation Flow

The BIST circuit compilation flow using BRAINS is given in Figure 27. The *memory specifications* and *test requirements* are provided via the user-interface. The memory specifications include the timing parameters, memory architecture (synchronous/asynchronous SRAM, single-port/multi-port SRAM and register file, EDO DRAM, SDRAM, 1T-SRAM, flash memory, etc.), memory configuration (data width, address width), etc. The test requirements include the test algorithm requirements (which affect the choice of the march elements and the programmability), address ordering (counting or pseudo-random, interleaved or non-interleaved), supported test modes (go/no-go test, burn-in test, diagnosis test), etc.

The interface of BRAINS is flexible—the user can generate the BIST circuit using the GUI (see Figure 28) or command shell, and evaluate the

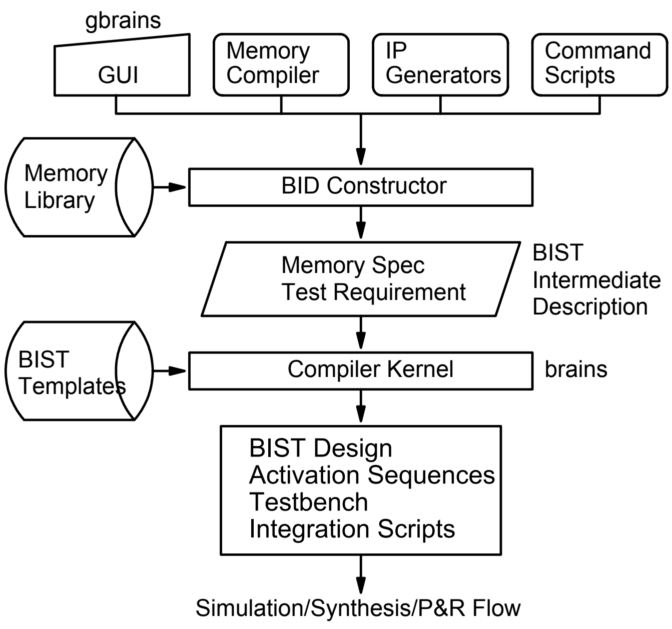
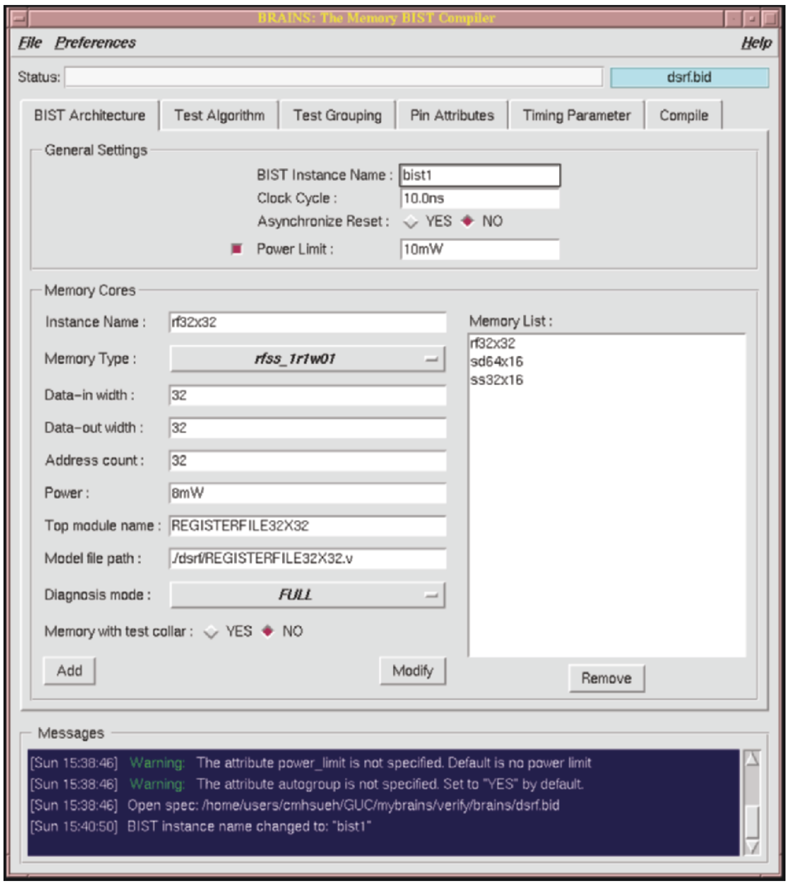


Figure 27. The BIST compilation flow using BRAINS

memory test efficiency among different designs easily. BRAINS can also be integrated with a memory compiler to deliver BISTed memory IPs.

The *BIST Intermediate Description* (*BID*) Constructor translates the user-defined parameters to an internal format with memory specifications and test requirements. BRAINS uses an object-oriented BID structure as shown in Figure 29, which is reusable and flexible for processing the memory specifications and test requirements. Future extension is straightforward. The BIST object consists of global parameters (clock rate, diagnosis requirement, etc.) and other sub-objects. The March object defines the test requirements, while the Schedule object records the grouping and scheduling information.
In this figure the dashed box defines a single memory object. Multiple Memory objects can be inherited or defined. Common memory specifications are predefined in the memory library. The user can access existing Memory objects and construct the target one with slightly modification. Using the presented BIST architecture as the template, the compiler generates the BIST design, control signals, and necessary scripts for synthesis and integration. The process can be integrated into an existing logic design flow easily. With TGS, test time can be further reduced, under certain given constraints. The rapid generation process makes the system handy at the early design phase of a system chip.



Status: dsrf.bid

General Settings

BIST Instance Name :	bist1
Clock Cycle :	10.0ns
Asynchronize Reset :	YES [✓ NO]
■ Power Limit :	10mW

Memory Cores

Instance Name :	rf32x32
Memory Type :	rfss_1r1w01
Data-in width :	32
Data-out width :	32
Address count :	32
Power :	8mW
Top module name :	REGISTERFILE32X32
Model file path :	./dsrf/REGISTERFILE32X32.v
Diagnosis mode :	FULL
Memory with test collar :	YES [✓ NO]

Memory List :
rf32x32
sd64x16
ss32x16

Messages

```
[Sun 15:38:46]  Warning:  The attribute power_limit is not specified. Default is no power limit
[Sun 15:38:46]  Warning:  The attribute autogroup is not specified. Set to "YES" by default.
[Sun 15:38:46]  Open spec: /home/users/cmhsueh/GUC/mybrains/verify/brains/dsrf.bid
[Sun 15:40:50]  BIST instance name changed to: "bist1"
```

Figure 28. The graphic user interface

After the BID format is generated, the user can also customize the BIST circuit by changing the description. The compiler kernel then parses the BID file and loads the BIST templates to generate the Controller, Sequencer, and Test Pattern Generator. The compiler engine configures the programmability of the BIST circuit and refines the memory access timing according to the timing specifications and test requirements. It generates the synthesizable RTL model for the BIST circuit, BIST activation sequence, test-bench, synthesis scripts, and the UNIX Makefile for integrated command-level operations. The synthesizable BIST model is in the Verilog format. The BIST activation sequence can be used to control the BIST from a simple tester interface. Different test algorithms can be applied during field test. The test-bench contains stimulus that can be used for behavior-level and gate-level simulations. Automatic synthesis can be done by a synthesis tool using the synthesis scripts. The generated logic

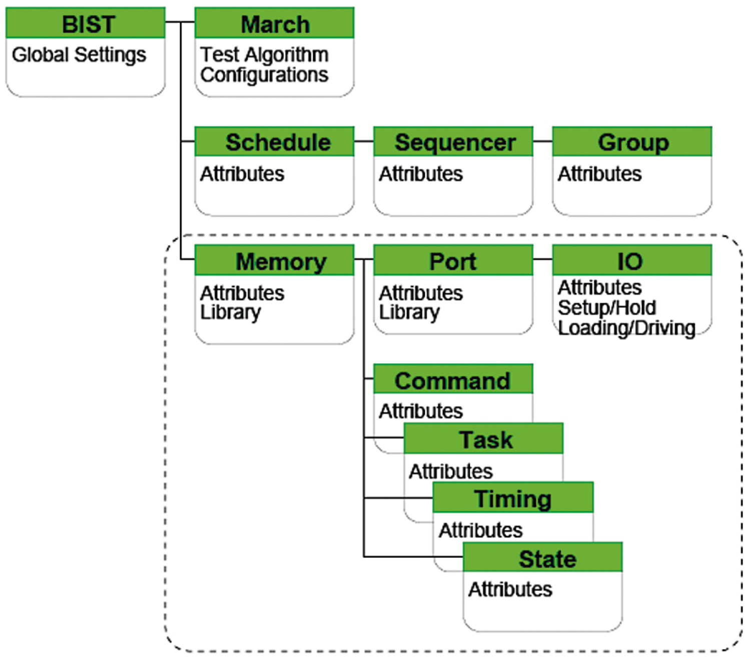


Figure 29. The BID architecture

circuit (in the netlist level) is then simulated and compared with the behavior-level result for design verification.

4. EXPERIMENTAL RESULTS

A test chip has been implemented and fabricated to verify the proposed approaches. The test time was analyzed and compared for both the session-based and non-session-based test scheduling methods. This test chip is an industrial digital still camera (DSC) chip, implemented with a standard 0.25μm CMOS technology. The digital part of the chip mainly includes a processor, JPEG codec, TV encoder, USB, external memory interface, and tens of single-port and two-port synchronous SRAMs with different sizes. Figure 30 gives the block diagram of this test chip. There are 37 input pins and 70 output pins.

The IPs to be wrapped in this test chip include the USB, TV encoder, and JPEG cores. The USB core has 4 clock domains, 3 reset signals, 1 SE signal, and 6 test signals. There are 4 scan chains with dedicated scan input and output for each clock domain. The TV encoder has both scan and functional.

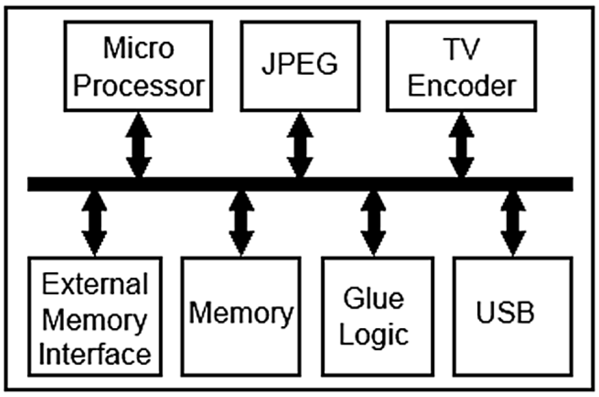


Figure 30. Block diagram of the test chip

tests. The test pins include one clock, reset, SE, and test enable signals There are two scan chains in the TV encoder, where one scan chain shares the output with a functional output. The JPEG core has only functional patterns and one clock domain. The clock signals for the IPs are generated by an internal PLL. The detailed test information of the IPs, including the number of test IOs (TI & TO), primary IOs (PI & PO), scan chains, and test patterns, is shown in Table 2. In the scan-test mode, the USB core has 2 scan vectors that are enabled by SE during Pulse-Clock, while the TV encoder has only one.

Table 2. Test information of the cores

Core	TI	TO	PI	PO	Scan chains (Lengths)	Patterns (Type)
USB	18	4	221	104	4 (1,629, 78, 293, 45)	716 (Scan)
TV	6	1	25	40	2 (577,576)	229 (Scan)
						202,673 (Func.)
JPEG	1	0	165	104	No scan	235,696 (Func.)

Figure 31 shows the functional-pattern waveforms of the TV encoder and JPEG cores. To apply the functional patterns for the TV encoder and JPEG cores, the updating control signal is delayed. For the TV encoder, the input WBR cells with Timing 1 waveform (see Figure 31) are controlled by the updating signals with 2-cycle delay. Other input WBR cells are with Timing 2 waveform, and are controlled by the updating signals with 1-cycle delay. Functional pattern application for the JPEG core is similar.

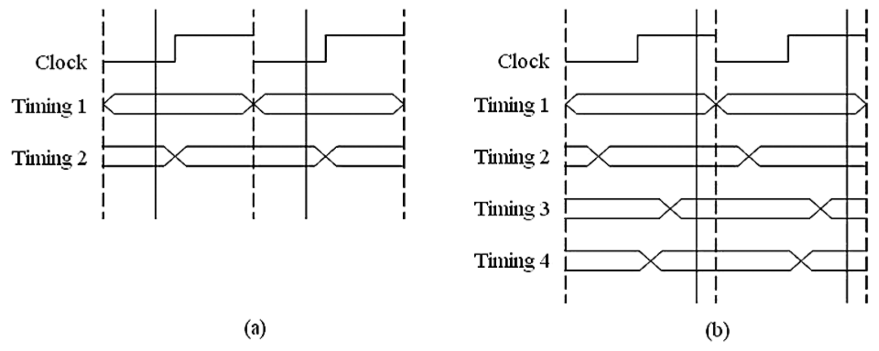


Figure 31. Functional-pattern waveforms of the (a) TV encoder and (b) JPEG cores

4.1 Test Time Analysis

In this section, we analyze the test time of the *session-based* and *non-session-based test scheduling* approaches under the IO resource constraints. In non-session-based test scheduling the cores can be scheduled at any time; while the session-based test scheduling groups the cores into several test sessions. All core tests in the same session start at the same time, and the next session cannot start until all tests in the current one finish. In our test chip, there are only 37 PIs and 70 POs. All test IOs share these chip IOs, and one additional pin is added to switch between the functional mode and test mode. Since we have more test inputs than test outputs, we discuss only the number of needed test inputs. We consider three cases under this IO limitation—Case A and Case B use the proposed session-based test scheduling approach under TACS, while Case C uses a non-session-based test schedulilng approach with dedicated WSC signals to control the core test wrappers.

For Cases A and B, the TRST signal of the IEEE 1149.1 TAP is also used to switch between the functional mode and test mode. When TRST=0, the chip is in functional mode. It is in test mode when TRST=1, where all test IOs are connected to the chip IO pins. The multiplexed pins are controlled by TRST. To reduce the test idle time during shifting, scan and functional tests are placed in different test sessions. The TAP requires 3 chip inputs—TCK, TMS, and TDI. In Case A, a coarse test scheduling is shown in Figure 32(a). The coarse test scheduling only partitions core tests into several test sessions, without TAM assignment. In this case, the needed test clock and test reset signals are dominated by Session 1. Five test clock signals (four clock signals for USB core and one for TV encoder) and four reset signals (three reset signals for USB and one for TV encoder) are,

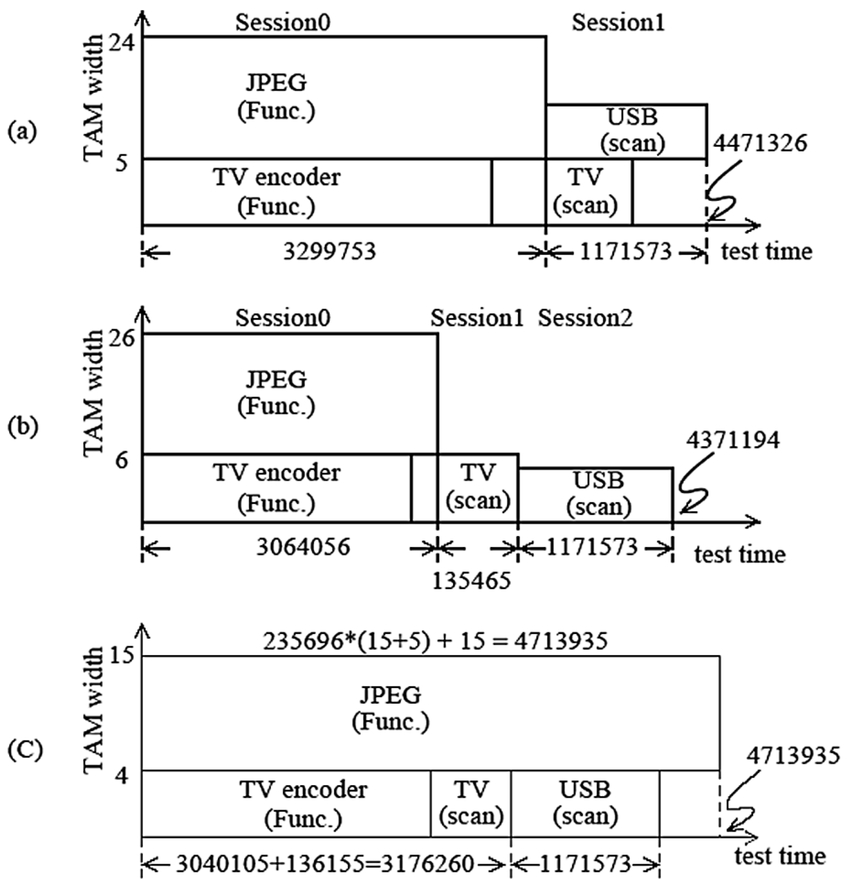


Figure 32. Scheduling results of the test chip: (a) Case A—session-based, with USB and TV encoder tested in parallel, (b) Case B—session-based, with USB and TV encoder tested in serial, and (c) Case C—non-session-based, with dedicated WSC signals

required. The test clock and test reset signals take 9 chip inputs. In addition the TSE takes one chip input to control the SE signals of all cores. Other test enable signals and WSC signals are generated by the Test Controller, which requires no chip input. The rest (37-3-9-1=24) of the chip inputs can be used as the TAM inputs. The second step of test scheduling assigns this TAM of width 24 to the IPs based on the coarse scheduling result. The final TAM assignment is shown in Figure 32(a). The numbers of clock cycles to loadand unload test data are 9 and 1629 for Session 0 and Session 1, respectively. From (1) and (2), the test time is about 4.4M clock cycles. The detailed test time figures are shown in Table 3.

Table 3. Test time comparison

Case A: session-based (Figure 32(a))	
Session0:	235,696 (9+5)+9=3,299,753
Session1:	716 (1,629+5)+1,629=1,171,573
Total:	3,299,753+1,171,573=4,471,326 (0.948)
Case B: session-based (Figure 32(b))	
Session0:	235,696 (8+5)+8=3,064,056
Session1:	229 (577+7+5)+(577+7)=135,465
Session2:	716 (1,629+5)+1,629=1,171,573
Total:	3,064,056+135,465+1,171,573=4,371,194 (0.927)
Case C: non-session-based (Figure 32(c))	
USB:	716 (1,629+5)+1,629=1,171,573
TV:	202,673 (10+5)+10+
	229 (587+50)+587=3,176,260
JPEG:	235,696 (15+5)+15=4,713,935
Total:	4,713,935 (1) (dominated by JPEG)

The difference between Case A and Case B is whether USB and TV encoder are tested concurrently or not. Case B is shown in Figure 32(b), where USB and TV encoder are tested sequentially, requiring only 4 test clock signals and 3 test reset signals. The test control IOs take only 11 chip inputs (TCK, TMS, TDI, TSE, 4 test clock signals, and 3 reset signals), and there are 26 chip inputs that can be used as the TAM inputs. The final TAM assignment is also shown in Figure 32(b), and the test time is about 4.3M clock cycles. The detailed test time figures are also shown in
Table *3*.

Both Case A and Case B are session-based test scheduling. When the test IO resource constraint is considered, parallel testing (Case A) may not be better than serial testing (Case B). This is because more test control IOs are needed for parallel testing, so fewer IO pins can be used as the test data IOs (i.e., TAM IOs). Since there are also cases when parallel testing leads to shorter test time than serial testing, it is important to take chip IO pins into consideration so far as test time evaluation is concerned.

We now discuss the test time of non-session-based test scheduling under the same IO resource constraint (Case C). The test time calculation follows the assumption as in previous works [12][13][14][15][10], i.e., 1) the core test time is the product of test vector count and the time to load-unload a test vector, and 2) each of the cores can be tested at any time. The coarse scheduling result of Case C is shown in Figure 32(c). In this case, the TAM IOs are partitioned into two groups, one for the JPEG core and another for the USB and TV encoder cores. Since the USB and TV encoder cores are not tested concurrently, they can share the same clock, reset, SE signals and

TAM inputs. One chip input is used to switch the shared signals between these two cores. Five chip inputs are used for the scan clocks, 3 chip inputs for test reset, and 1 chip input for the SE signal. However, to test the JPEG and TV encoder (or USB) cores concurrently, 2 groups of WSC signals are needed. One is for JPEG and another is for TV encoder and USB. Note that WRCK, and WRSTN can be shared by these two groups, but other signals can not. A total of 12 WSC signals are needed. Only 15 chip inputs can be used as TAM inputs in this case. The TAM assignment result also is shown in Figure 32(c). The test time comparison of the three cases is given in

Table *3*. The test time of Case C is 4.7M clock cycles, which is obviously longer than either Case A or Case B, i.e., the session-based approaches. The normalized total test time for each case is also shown in

Table *3*, inside the parentheses. Case B has more than 7% test time improvement over Case C. Although non-session-based test scheduling has no idle time that appears in the results of session-based test scheduling, it requires more test control IOs. This reduces the number of TAM IOs, and may lead to longer test time. The proposed session-based approach and test controller design use fewer test control IOs, so more test IOs can be allocated for the test data (as TAM IOs), resulting in shorter test time.

Table *4* summarizes the numbers of test inputs used in the three schedules. The total test IOs of these three cores are 19, including 6 clock signals, 4 reset signals, 7 test enable signals, and 2 SE signals. With shared test IOs, the test control IO counts are reduced to 10, 8, and 9 for Case A, Case B, and Case C, respectively. This also shows that the proposed test IO reduction method can reduce the number of test control IOs, and thus the test cost.

Table 4. Numbers of test inputs for the 3 cases

	Switch	TAP	Shared Test IO			Dedicated WSC	TAM
			Clock	Reset	SE		
Case A	No	3	5	4	1	No	24
Case B	No	3	4	3	1	No	26
Case C	1	No	5	3	1	12	15

4.2 Area Overhead and Test Result

With STEAC, the Test Wrappers, TAM, and Test Controller have been automatically generated and inserted into the original test chip design in 5 minutes, using a SUN Blade 1000 workstation with dual 750MHz processors and 2GB RAM. The area of the WBR cell, WC_SD1_CII_UD, is equivalent to 26 two-input NAND gates. The Test Controller and TAM multiplexer require about 371 and 132 gates, respectively—their hardware overhead is only about 0.3%. The total hardware overhead is 14.08% as shown in Table 5. The overhead is dominated by the WBR cells, which are dedicated cells in this experiment. The overhead can be reduced by, e.g., sharing the WBR and core IO registers or customizing the WBR cell design.

Table 5. Area overhead of the test circuitry

	USB	TV	JPEG	TACS	TAM	Total
Original	34,781	21,554	70,794	—	—	127,129
Test Circuit	8,580	1,716.0	7,101.6	371	132	17397.6
Overhead	24.67%	7.96%	10.03%	—	—	14.08%

4.3 Experimental Results for BRAINS

Based on BRAINS, Table 6 gives the area overhead for a test chip, which consists of 6 single-port SRAM cores, using a 0.15μm CMOS process technology. The BIST design supports diagnosis and programmable test algorithm, and has a built-in *March CW* word-oriented test algorithm [46][50]. With shared Controller and Sequencer, the overhead is about 2% and the test time is reduced by 59% (the test time is 667,648 clock cycles, as compared with 1,641,472 when testing the memories serially).

Table 6. The BIST area overhead for a test chip

Memory Config.	BIST (μm^2)				Memory (μm^2)	Area Overhead
	Ctrl.	Seq.	TPG	Total		
16K × 32	4541.96	6514.73	11905.44	107378	5301800	2.025%
12K × 24			9443.30			
8K × 64			23099.77			
4K × 128			41389.96			
1K × 8			3930.25			
512x16			6530.03			

5. CONCLUSIONS

We have presented a practical session-based test scheduling model considering IO resource constraints. It results in shorter test time than non-session-based test scheduling. The test cost and test time can be further reduced with test IO reduction. We also have presented an improved Test Wrapper architecture. With the proposed WBR cells and enhanced TAP Controller, the IEEE 1500 Test Wrapper supports both scan and functional tests, even when the scan and functional IOs are shared. The complex timing of the functional patterns can be applied through our Test Controller easily. As a result, this work solved the major issues of practical SOC test integration. Without addressing these issues, the test scheduling, test control, Test Wrapper and TAM architecture design would have been impractical. The test integration platform, STEAC, has been used to develop an industrial SOC design. From the experimental results, we have shown that the test scheduling and test IO reduction effectively lead to shorter test time. The area overhead for Test Controller and TAM is about 0.3% in the case, justifying the effectiveness of the approach.

For embedded memories, BRAINS generates the synthesizable RTL code for the BIST circuit in Verilog, as well as its activation sequence, test-bench, and synthesis scripts. It supports programmable march tests. The improved BIST architecture extends the ability of system-level test integration, multi-port and multi-memory support, and test grouping and scheduling for parallel testing. It provides at-speed testing and diagnosis of the RAM under test. BRAINS can be used for a wide range of RAM architectures and configurations. Using the OCB interface, the test control and observation for diagnostics are simple and flexible, which is especially important in an SOC design. The BIST access can be parallel for easy test control, or serial for simple IO interface, depending on the test requirement. The proposed TGS algorithm facilitates test grouping and scheduling at the system level under various user-defined constraints. Future extension can easily be implemented. The flexibility is fulfilled by using the novel BID format.

REFERENCES

[1] Y. Zorian, E. J. Marinissen, and S. Dey, “Testing embedded-core-based system chips”, *IEEE Computer*, vol. 32, no. 6, pp. 52–60, June 1999.

[2] C.-W. Wu, J.-F. Li, and C.-T. Huang, “Core-based system-on-chip testing: Challenges and opportunities”, *J. Chinese Institute of Electrical Engineering*, vol. 8, no. 4, pp. 335–353, Nov. 2001.

[3] E. Marinissen, R. Kapur, and Y. Zorian, “On using IEEE 1500 SECT for test plug-n-play”, in *Proc. Int. Test Conf. (ITC)*, 2000, pp. 770–777.

[4] IEEE, *IEEE Standard Testability Method for Embedded Core-based Integrated Circuits*, IEEE Standards Department, Piscataway, Aug. 2005.

[5] IEEE, *IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data*, IEEE Standards Department, Piscataway, Sept. 1999.

[6] R. Kapur, M. Lousberg, T. Taylor, B. Keller, P. Reuter, and D. Kay, “CTL: the language for describing core-based test”, in *Proc. Int. Test Conf. (ITC)*, 2001, pp. 131–139.

[7] M. Benabdenbi, W. Maroufi, and M. Marzouki, “CAS-BUS: a scalable and reconfigurable test access mechanism for systems on a chip”, in *Proc. Design, Automation and Test in Europe (DATE)*, Paris, Mar. 2000, pp. 141–145.

[8] L. Whetsel, “Addressable test ports—an approach to testing embedded cores”, in *Proc. Int. Test Conf. (ITC)*, 1999, pp. 1055–1061.

[9] E. J. Marinissen and S. K. Goel, “Analysis of test bandwidth utilization in test bus and TestRail architectures for SOCs”, in *Proc. IEEE Int. Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, Apr. 2002, pp. 52–60.

[10] H.-S. Hsu, J.-R. Huang, K.-L. Cheng, C.-W. Wang, C.-T. Huang, C.-W. Wu, and Y.-L. Lin, “Test scheduling and test access architecture optimization for system-on-chips”, in *Proc. 11th IEEE Asian Test Symp. (ATS)*, Guam, Nov. 2002, pp. 411–416.

[11] A. Benso, S. Di Carlo, P. Prinetto, and Y. Zorian, “A hierarchical infrastructure for SoC test management”, *IEEE Design & Test of Computers*, vol. 20, no. 4, pp. 32–39, Jul.-Aug. 2003.

[12] K. Chakrabarty, “Test scheduling for core-based systems using mixed-integer linear programming”, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 10, pp. 1163–1174, Oct. 2000.

[13] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, “Test wrapper and test access mechanism co-optimization for system-on-chip”, *J. Electronic Testing: Theory and Applications*, vol. 18, pp. 213–230, Apr. 2002.

[14] C.-P. Su and C.-W. Wu, “Graph-based power-constrained test scheduling for SOC”, in *Proc. IEEE Int. Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, Brno, Czech Republic, Apr. 2002, pp. 61–68.

[15] S. K. Goel and E. J. Marinissen, “Effective and efficient test architecture design for SOCs”, in *Proc. Int. Test Conf. (ITC)*, Baltimore, Oct. 2002, pp. 529–538.

[16] C.-W. Wang, J.-R. Huang, K.-L. Cheng, H.-S. Hsu, C.-T. Huang, C.-W. Wu, and Y.-L. Lin, “A test access control and test integration system for system-on-chip”, in *Proc. Sixth IEEE Int. Workshop on Testing Embedded Core-Based System-Chips (TECS)*, Monterey, California, May 2002, pp. P2.1–P2.8.

[17] J.-F. Li, H.-J. Huang, J.-B. Chen, C.-P. Su, C.-W. Wu, C. Cheng, S.-I Chen, C.-Y. Hwang, and H.-P. Lin, “A hierarchical test methodology for system-on-chip”, *IEEE Micro*, vol. 22, no. 5, pp. 69-81, Sept./Oct. 2002.

[18] K.-L. Cheng, J.-R. Huang, C.-W. Wang, C.-Y. Lo, L.-M. Denq, C.-T. Huang, C.-W. Wu, S.-W. Hung, and J.-Y. Lee, “An SOC test integration platform and its industrial realization”, in *Proc. Int. Test Conf. (ITC)*, Charlotte, Oct. 2004.

[19] IEEE, IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture, IEEE Standards Department, Piscataway, May 1990.

[20] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, “ITC’02 SOC test benchmarks”, http://www.extra.research.philips.com/itc02socbenchm/, 2002.

[21] E. J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, and L. Whetsel, “Towards a standard for embedded core test: An example”, in *Proc. Int. Test Conf. (ITC)*, 1999, pp. 616–626.

[22] J. Aerts and E. J. Marinissen, “Scan chain design for test time reduction in core-based ICs”, in *Proc. Int. Test Conf. (ITC)*, 1998, pp. 448–457.

[23] P. Varma and S. Bhatia, “A structured test reuse methodology for core-based system chips”, in *Proc. Int. Test Conf. (ITC)*, 1998, pp. 294–302.

[24] P. Harrod, “Testing reusable IP—a case study”, in *Proc. Int. Test Conf. (ITC)*, 1999, pp. 493–498.

[25] E. J. Marinissen, R. Arendsen, and G. Bos, “A structured and scalable mechanism for test access to embedded reusable cores”, in *Proc. Int. Test Conf. (ITC)*, 1998, pp. 284–293.

[26] M. Benabdenbi, W. Maroufi, and M. Marzouki, “Testing TAPed cores and wrapped cores with the same test access mechanism”, in *Proc. Design, Automation and Test in Europe (DATE)*, Munich, Mar. 2001, pp. 150–155.

[27] K. Chakrabarty, “Design of system-on-a-chip test access architecture using integer linear programming”, in *Proc. IEEE VLSI Test Symp. (VTS)*, 2000, pp. 127–134.

[28] E. Larsson and Z. Peng, “An integrated system-on-chip test framework”, in *Proc. Design, Automation and Test in Europe (DATE)*, Munich, Mar. 2001, pp. 138–144.

[29] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, “Test wrapper and test access mechanism cooptimization for system-on-chip”, in *Proc. Int. Test Conf. (ITC)*, Baltimore, Oct. 2001, pp. 1023–1032.

[30] Y. Huang, W.-T. Cheng, C.-C. Tsai, N. Mukherjee, O. Samman, Y. Zaidan, and S. M. Reddy, “Resource allocation and test scheduling for concurrent test of core-based SOC design”, in *Proc. Tenth IEEE Asian Test Symp. (ATS)*, Kyoto, Nov. 2001, pp. 265–270.

[31] L. Whetsel, “A IEEE 1149.1 base test access architecture for ICs with embedded cores”, in *Proc. Int. Test Conf. (ITC)*, 1997, pp. 69–78.

[32] IEEE, *IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture*, IEEE Standards Department, Piscataway, May 1990.

[33]K. K. Saluja, S. H. Sng, and K. Kinoshita, “Built-in self-testing RAM: A practical alternative”, *IEEE Design & Test of Computers*, vol. 4, no. 1, pp. 42–51, Feb. 1987.

[34]R. Dekker, F. Beenker, and L. Thijssen, “A realistic self-test machine for static random access memories”, in *Proc. Int. Test Conf. (ITC)*, 1988, pp. 353–361.

[35]A. J. van de Goor, *Testing Semiconductor Memories: Theory and Practice*, John Wiley & Sons, Chichester, England, 1991.

[36]B. F. Cockburn and Y.-F. Nicole Sat, “Synthesized transparent BIST for detecting scrambled patternsensitive faults in RAMs”, in *Proc. Int. Test Conf. (ITC)*, Oct. 1995, pp. 23–32.

[37]J. Dreibelbis, J. Barth, H. Kalter, and R. Kho, “Processor-based built-in self-test for embedded DRAM”, *IEEE Journal of Solid-State Circuits*, pp. 1731–1740, Nov. 1998.

[38]C.-W. Wu, “Testing embedded memories: Is BIST the ultimate solution?”, in *Proc. Seventh IEEE Asian Test Symp. (ATS)*, Singapore, Dec. 1998, pp. 516–517.

[39]C.-T. Huang, J.-R. Huang, C.-F. Wu, C.-W. Wu, and T.-Y. Chang, “A programmable BIST core for embedded DRAM”, *IEEE Design & Test of Computers*, vol. 16, no. 1, pp. 59–70, Jan.-Mar. 1999.

[40]O. Kebichi and M. Nicolaidis, “A tool for automatic generation of BISTed and transparent BISTed RAMs”, in *Proc. IEEE Int. Conf. Computer Design (ICCD)*, Oct. 1992, pp. 570–575.

[41]R. Rajsuman, “RAMBIST builder: A methodology for automatic built-in self-test design of embedded RAMs”, in *Proc. IEEE Int. Workshop on Memory Technology, Design and Testing (MTDT)*, 1996, pp. 50–56.

[42]M. Marinescu, “Simple and efficient algorithms for functional RAM testing”, in *Proc. Int. Test Conf. (ITC)*, 1982, pp. 236–239.

[43]K. Zarrineh and S. J. Upadhyaya, “Programmable memory BIST and a new synthesis framework”, in *Proc. Int. Symp. Fault Tolerant Computing (FTCS)*, Montreal, June 1999, pp. 352–355.

[44]K. Zarrineh and S. J. Upadhyaya, “On programmable memory built-in self test architecutres”, in *Proc. Design, Automation and Test in Europe (DATE)*, Paris, Mar. 1999, pp. 708–713.

[45]K.-J. Lin and C.-W. Wu, “PMBC: a programmable BIST compiler for memory cores”, in *Third IEEE Int. Workshop on Testing Embedded Core-Based System-Chips*, Dana Point, Apr. 1999, pp. P2.1–P2.6.

[46]C.-F. Wu, C.-T. Huang, K.-L. Cheng, and C.-W. Wu, “Simulation-based test algorithm generation for random access memories”, in *Proc. IEEE VLSI Test Symp. (VTS)*, Montreal, Apr. 2000, pp. 291–296.

[47]C. Cheng, C.-T. Huang, J.-R. Huang, C.-W. Wu, C.-J. Wey, and M.-C. Tsai, “BRAINS: A BIST complier for embedded memories”, In *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 299–307, Yamanashi, Oct. 2000.

[48]K.-L. Cheng, C.-M. Hsueh, J.-R. Huang, J.-C. Yeh, C.-T. Huang, and C.-W. Wu, “Automatic generation of memory built-in self-test cores for system-on-chip”, in *Proc. Tenth IEEE Asian Test Symp. (ATS)*, Kyoto, Nov. 2001, pp. 91-96.

[49]D. Flynn, “AMBA: Enabling reusable on-chip designs”, *IEEE Micro*, 17(4):20–27, July/Aug. 1997.

[50]C.-F. Wu, C.-T. Huang, and C.-W. Wu, “RAMSES: a fast memory fault simulator”, In *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 165–173, Albuquerque, Nov. 1999.

Chapter 9

PHYSICAL DESIGN FOR SYSTEM-ON-A-CHIP

Yao-Wen Chang[1], Tung-Chieh Chen[2], and Huang-Yu Chen[2]
[1]Graduate Institute of Electronics Engineering, National Taiwan University
[2]Department of Electrical Engineering and Graduate Institute of Electronics Engineering, National Taiwan University
Emails: ywchang@cc.ee.ntu.edu.tw; {donnie, yellowfish}@eda.ee.ntu.edu.tw

1. INTRODUCTION

This chapter is focused on the physical design for system-on-a-chip (SOC). Physical design refers to all synthesis steps that convert a circuit representation (gates, transistors) into a geometric representation (polygons and theirs shapes). See Figure 1 for an illustration. The geometric representation, also called *layout*, is used to design masks and then manufacture a chip. As a very complicated design process, modern physical design is typically divided into three major steps: *floorplanning*, *placement*, and *routing*. Floorplanning is an essential design step for hierarchical, building block design methodology. Given a set of hard blocks (whose shapes cannot be changed) and/or soft blocks (whose shapes can be adjusted) and a netlist, floorplanning determine the shapes of soft blocks and assemble the blocks into a rectangle (chip) such that a predefined cost metric (such as the chip area, wirelength, wire congestion) is optimized. Placement is the process of assigning the circuit components into a chip region. It can be considered as a restricted floorplanning problem for hard blocks with some dimension similarity. Following placement, the routing process defines the precise paths for conductors that carry electrical signals on the chip layout to interconnect all pins that are

S.Y.-L. Lin (ed.), Essential Issues in SOC Design, 311–403.
© 2006 *Springer.*

electrically equivalent. After routing, some physical verification processes (such as design rule checking (DRC), performance checking, and reliability checking) are performed to verify if all geometric patterns, circuit timing, and electrical effects satisfy the design rules and specifications.

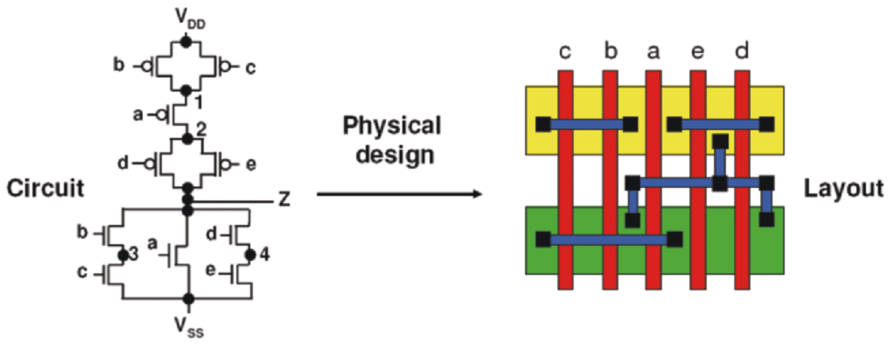


Figure 1. The function of physical design

With the continued improvement of the nanometer IC technologies, modern VLSI designers can integrate a whole system with large-scale logic/functional blocks (e.g., multimedia blocks, communication blocks, microprocessors, embedded memory) in a single chip, and interconnect those blocks to provide high capability and flexibility for different needs. This is called *system-on-a-chip* (*SOC*) design. See Figure 2 for an example of SOC architecture.

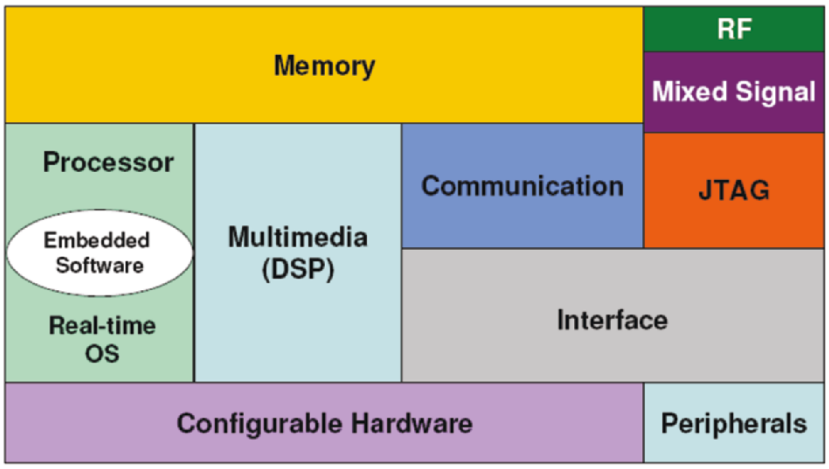


Figure 2. A typically SOC system

As the technology advances at a breathtaking speed, feature sizes and voltage levels are decreasing while die sizes, operating frequency, design complexities, and packing density are increasing, all drastically for modern SOC design. Physical design for such a system needs to consider the integration of large-scale digital and analog (mixed-signal) circuit blocks, the design of system interconnections/buses, and the optimization of circuit performance, area, power consumption, and signal and power integrity. On one hand, designs with billion transistors are already in production, IP blocks are widely reused, and a large number of buffer blocks are used for delay optimization as well as noise reduction in nanometer interconnect-driven design, which all drive the need of a modern physical design tool to handle large-scale designs. On the other hand, the highly competitive IC market requires faster design convergence, faster incremental design turnaround, and better silicon area utilization. Efficient and effective design methodology and tools capable of handling optimizing large-scale blocks are essential for modern SOC designs.

In this chapter, we focus on the recent development on floorplanning, placement, and routing, with special treatments on the impacts of the modern SOC design on these design steps. Specifically, we introduce the state-of-the-art design algorithms, frameworks, and methodology for handling the design complexity, timing closure, and signal/power integrity arising from modern SOC designs for faster design convergence.

2. FLOORPLANNING

2.1 Introduction

Floorplanning is an essential design step for hierarchical, building-block design methodology. Floorplanning gives early feedback that suggests architectural modifications, estimates the chip area, and estimates delay and congestion due to wiring. As technology advances, design complexity is increasing and the circuit size is getting larger. To cope with the increasing design complexity, hierarchical design and IP blocks are widely used. This trend makes floorplanning much more critical to the quality of a VLSI design than ever.

An SOC design often consists of large-scale functional blocks. Designs with billions of transistors are even already in production. To cope with the increasing design complexity, IP blocks are widely reused for large-scale designs. Therefore, efficient and effective design methodology and tools capable of placing and optimizing large-scale blocks are essential for modern chip designs. The floorplanning frameworks are evolving to tackle the challenges with constantly increasing design complexity. Three major frameworks have been extensively studied in the literature: the flat, hierarchical, and multilevel frameworks.

In this section, we first introduce two most popular floorplan representations, B*-tree [17] and Sequence Pair [88]. Then, three types of floorplanning frameworks are introduced in Section 2.4. Two important issues for modern SOC floorplanning, substrate noise and bus planning, are discussed in Section 2.5 and Section 2.6, respectively.

2.2 Problem Definition

To make this chapter self-contained, we shall start with the definition of the floorplanning problem. Let $B = \{b_1, b_2, ..., b_m\}$ be a set of m rectangular blocks whose respective width, height, and area are denoted by w_i, h_i, and a_i, $1 \le i \le m$. Each block is free to rotate. Let (x_i, y_i) denote the coordinate of the bottom-left corner of block b_i, $1 \le i \le m$, on a chip. A floorplan ***P*** is an assignment of (x_i, y_i) for each b_i, $1 \le i \le m$, such that no two blocks overlap. The goal of floorplanning is to optimize a predefined cost metric such as a combination of the area (i.e., the minimum bounding rectangle of ***P***) and wirelength (i.e., the summation of half bounding box of interconnections) induced by a placement.

2.3 Floorplanning Representations

We introduce the B*-tree [17] and Sequence Pair [88] floorplan representations, which are generally considered as the two most popular representations [14].

2.3.1 B*-tree

B*-trees are based on ordered binary trees and the admissible placement [48], for which no block can be moved to bottom and left, i.e., a bottom-left compacted placement. Inheriting from the nice properties of ordered binary trees, B*-trees are very easy for implementation and can perform the respective primitive tree operations search, insertion, and deletion in only constant, constant, and linear times. There exists a one-to-one correspondence between an admissible placement and its induced B*-tree; further, the transformation between them takes only amortized linear time.

Given an admissible placement, we can represent it by a unique B*-tree T. (See Figure 3(b) for the B*-tree representing the placement of Figure 3(a).) A B*-tree is an ordered binary tree whose root corresponds to the block on the bottom-left corner. Similar to the DFS procedure, we construct the B*-tree T for an admissible placement ***P*** in a recursive fashion: Starting from the root, we first recursively construct the left subtree and then the right subtree. Let R_i denote the set of blocks located on the right-hand side and adjacent to b_i. The left child of the node n_i corresponds to the lowest block in R_i that is unvisited. The right child of n_i represents the lowest block located above and with its x-coordinate equal to that of b_i.

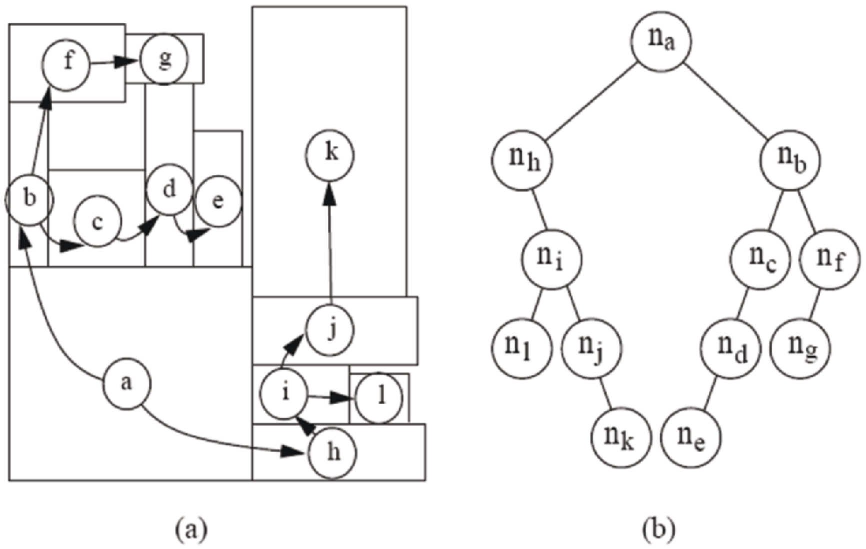


Figure 3. (a) An admissible placement. (b) The B*-tree representing the placement

Given a B*-tree T, we shall compute the x- and y-coordinates for each block associated with a node in the tree. Since the root of T represents the bottom-left block, the x- and y-coordinates of the block associated with the root $(x_{root}, y_{root}) = (0, 0)$. The B*-tree keeps the geometric relationship between two blocks as follows. If node n_j is the left child of node n_i, block b_j must be located on the right-hand side and adjacent to block b_i in the admissible placement; i.e., $x_j = x_i + w_i$. Besides, if node n_j is the right child of n_i, block b_j must be located above, with the x-coordinate of b_j equal to that of b_i; i.e., $x_j = x_i$. Therefore, given a B*-tree, the x-coordinates of all blocks can be determined by traversing the tree once in linear time.

To efficiently compute the y-coordinate from a B*-tree, the *contour* data structure presented in [48] is used to facilitate the operations on blocks. The contour structure is a doubly linked list for blocks, describing the contour curve in the current compaction direction by bookkeeping the x-range of each block and its corresponding y-coordinate of the top boundary. A *horizontal contour* (see Figure 4) can be used to reduce the running time for finding the y-coordinate of a newly inserted block. Without the contour, the running time for determining the y-coordinate of a newly inserted block would be linear to the number of blocks. By maintaining the contour structure, however, the y-coordinate of a block can be computed in amortized $O(1)$ time [48], resulting in an overall packing evaluation of amortized linear time. Figure 4 illustrates how to update the horizontal contour after inserting a new block.

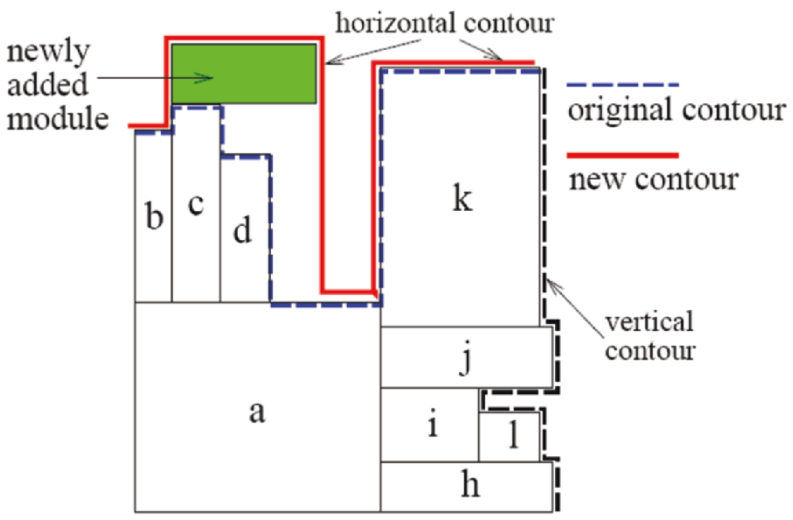


Figure 4. Adding a new block on the top, we search the horizontal contour from left to right and update it with the top boundary of the new block

2.3.2 Sequence pair

Sequence Pair (SP) [88] uses an ordered pair of block name sequences to model a general floorplan. Given an SP (Γ_+, Γ_-), blocks b_i and b_j are related in exactly one of four ways: b_j is after/before b_i in (Γ_+, Γ_-). The geometric relation of blocks can be derived from an SP as follows. Block b_i is left (right) to block b_j if b_i appears before (after) b_j in both Γ_+ and Γ_-. Block b_i is below (above) block b_j if b_i appears after (before) b_j in Γ_+ and b_i appears before (after) b_j in Γ_-.

It is easily seen that the constraint imposed on the packing by a sequence pair is unique, and the constraint is always satisfiable. We can consider an $m \times m$ grid. Label the horizontal and vertical grid lines with block names along Γ_+ and Γ_- from top and from left, respectively. A cross point of the horizontal grid line of label i and the vertical grid line of label j is referred to by (i, j). Then, rotate the resultant grid by 45 degrees counter-clockwise to get an oblique grid. (See Figure 5.) Put each block b_i with its center being on (i, i). Expand the separation of grid lines enough to eliminate overlapping of blocks. (The expansion is enough if the separation is $\sqrt{2}$ times larger than the longest width/height over blocks.) The resultant packing trivially satisfies the constraint implied by the given SP. An example with $(\Gamma_+, \Gamma_-) = (ecadfb, fcbead)$ is shown in Figure 5.

Given (Γ_+, Γ_-), one of the optimal packing under the constraint can be obtained in $O(n^2)$ time by applying the well-known *longest-path algorithm* on a node-weighted directed acyclic graphs with n nodes. The process is given below.

Based on "left of" constraint of (Γ_+, Γ_-), a directed and node-weighted graph $G_H(V, E)$, where V is the node set and E is the edge set, called the *horizontal-constraint graph*, is constructed as follows.

- V : source s, sink t, and m nodes labelled with block names.
- E: (s,i) and (i,t) for each block b_i, and (i, j) if and only if b_i appears before (after) b_j in both Γ_+ and Γ_- ("left of" constraint).
- Node-weight: zero for s and t, width of block b_i for the other nodes.

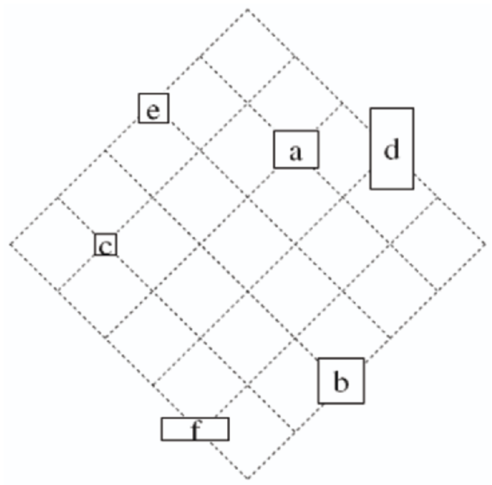


Figure 5. A packing on an oblique grid for $(\Gamma_+, \Gamma_-) = (ecadfb, fcbead)$

Similarly, the *vertical-constraint graph* $G_V(V,E)$ is constructed using the "below" constraint and the height of each block.

Neither of these graphs contains any directed cycle. We set the x-coordinate of b_i to be the longest-path length from s to i in G_H. The y-coordinate of b_i is set independently using G_V. If two blocks b_i and b_j have a horizontal relation, then there is an edge between i and j in G_H to guarantee that they do not overlap horizontally in the resultant placement. Similarly, if b_i and b_j have a vertical relation, they do not overlap vertically. Thus no two blocks overlap each other in the resultant placement because any pair of blocks are either in horizontal or vertical relation.

The width and the height of the chip is determined by the longest-path length between the source and the sink in G_H and G_V, respectively. Since the width and the height of the chip is independently minimized, the resultant packing is the best of all the packings under the constraint. The longest-path length calculation on each graph can be done in $O(n^2)$ time, where n is the number of nodes in the graph. The packing time of Sequence Pair can be reduced to $O(n\lg\lg n)$ time by resorting to the longest common subsequence formulation [102].

As an example, G_H and G_V are shown in Figure 6 for $(\Gamma_+,\Gamma_-)=(ecadfb, fcbead)$. The resultant placement after the longest path length calculation is shown in Figure 7.

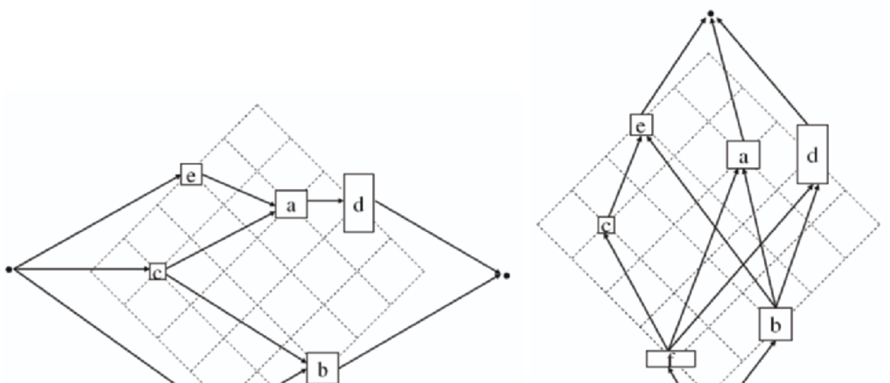


Figure 6. (a) The horizontal constraint graph G_H. (b) The vertical constraint graph G_V. (Transitive edges are not shown in both graphs for simplicity)

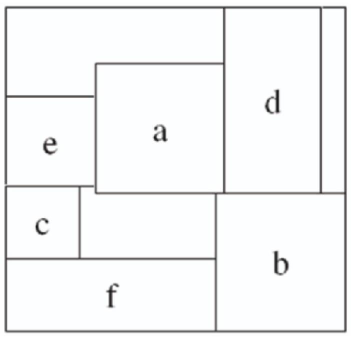


Figure 7. A best packing under the constraint implied by $(\Gamma_+, \Gamma_-) = (ecadfb, fcbead)$

2.4 Floorplanning Frameworks

To handle large-scale building block designs for SOCs, directly applying simulated annealing to find a good floorplan is often not feasible. Thus, multilevel approaches are applied to handle large-scale floorplanning. In the following, we first introduce the flat floorplanning framework, and then describe the multilevel floorplanning approaches.

2.4.1 Flat approaches

Simulated annealing (SA for short) [71] is widely used for floorplan optimization. It is an optimization scheme with non-zero probability for accepting inferior (uphill) solutions. The probability depends on the difference of the solution quality and the temperature. The probability is defined as follows:

$$\Pr ob(S \to S') = \begin{cases} 1 & if \quad \Delta C \le 0 \\ \min\{1, e^{-\Delta C/T}\} & if \quad \Delta C > 0 \end{cases} \tag{1}$$

where ΔC is the difference of the cost of the new solution and that of the current solution, and T is the current temperature. Thus, at a very high temperature, say $T \to \infty$, the probability approaches 1. In contrast, when $T \to 0$, the probability $e^{-\Delta C/T}$ approaches 0.

There are four basic ingredients for SA: solution space, neighborhood structure, cost function, and annealing schedule. Here, we use the B*-tree representation as an example to model a floorplan. The solution space consists of all B*-trees with the given nodes (blocks). To find a neighboring solution, we perturb a B*-tree to get another B*-tree by the following operations:

- Op1: Rotate a block.
- Op2: Move a node/block to another place.
- Op3: Swap two nodes/blocks.

For Op1, we rotate a block for a B*-tree node. For Op2, we delete a node and move it to another place in the B*-tree. For Op3, we swap two nodes in the B*-tree. After packing for the B*-tree, we obtain a new floorplan. Whether or not we take the new solution depends on the aforementioned probability which depends on its cost function. The cost function is defined based on problem requirements. For example, we may adopt the following cost function to optimize the wirelength and the area of a floorplan:

$$Cost = \alpha A + (1-\alpha)W, \tag{2}$$

where A is the current area, W is the current wirelength, and the user-specified α controls the weights for area and wirelength.

To determine the initial temperature T, a sequence of random moves are performed, and the average cost change Δ_{avg} for all uphill moves is computed. Then, the initial temperature T can be determined by $\Delta_{avg}/\ln P$, where P is the initial probability of accepting an uphill move and is set very close to 1 (say, 0.9). We obtain another floorplan by perturbing the B*-tree. If the new floorplan is better than the current one, we simply take it. On the other hand, if the new floorplan is worse, we accept it with a non-zero probability according to the current temperature.

The best results can be obtained when the floorplan achieves "equilibrium" at each value of T of the annealing process [96]. The "stopping criterion" is satisfied when the value of the cost function remains the same after several stages of the annealing process. The updating function for T is given below:

$$T_{new} = \lambda T_{old},\ 0 < \lambda < 1. \tag{3}$$

In the classical annealing schedule, the λ value is set to a fixed value [92]. A recommended value of λ is λ = 0.85. For better results, TimberWolf [96] set the initial λ to 0.8. The value of λ is gradually increased from its lowest value to its highest value (approximately 0.95), and is then gradually decreased back to its lowest value.

Chen and Chang propose a Fast Simulated Annealing (Fast-SA) process to integrate the random search with hill climbing more efficiently [22].

Unlike the classical SA [71] and the TimberWolf SA [96], the fast annealing process consists of three stages: (1) The high-temperature random search stage, (2) the pseudo-greedy local search stage, and (3) the hill-climbing search stage. At the first stage, it lets the temperature $T \to \infty$ so that the probability of accepting an inferior solution approaches 1. The process is like a random search to find the best solution. At the second stage, it makes $T \to 0$. Since the temperature is very low, it only accepts a very small number of inferior solutions, which is like a greedy local search. The third stage is the hill-climbing search stage. The temperature raises again to facilitate the hill climbing. Thus, it can escape from the local minimum and search for better solutions. The temperature reduces gradually, and very likely it finally converges to a globally optimal solution.

```
Algorithm: Simulated_Annealing_Floorplanning($F$)
Input: An initial floorplan $F$.
Output: A final floorplan.
1  $T \leftarrow \frac{\Delta_{avg}}{\ln P}$;   /* compute the initial temperature */
2  do
3      do
4          $F' = $ perturb($F$);   /* perturb the floorplan $F$ */
5          if (accept(cost($F$), cost($F'$))) then $F \leftarrow F'$;
6      until ("inner loop criterion" is satisfied);
7      $T = $ update($T$);
8  until ("stopping criterion" is satisfied);
9  return $F$.
```

Figure 8. Floorplanning using simulated annealing

2.4.2 The Λ-shaped multilevel floorplanning

The Λ-shaped multilevel framework adopts a two-stage technique, bottom-up coarsening followed by top-down uncoarsening. We take MB*-tree [76] for an example. Figure 9 illustrates the MB*-tree based Λ-shaped multilevel framework.

The clustering stage iteratively groups a set of (primitive or cluster) blocks (say, two blocks) based on a cost metric defined by area utilization, wirelength, and connectivity among blocks, and at the same time establishes the geometric relations among the newly clustered blocks by constructing a corresponding B*-subtree. The clustering procedure repeats until a single cluster containing all blocks is formed (or the number of blocks is smaller than a predefined threshold), denoted by a one-node B*-tree that bookkeeps the entire clustering scheme. We shall first consider the clustering metric.

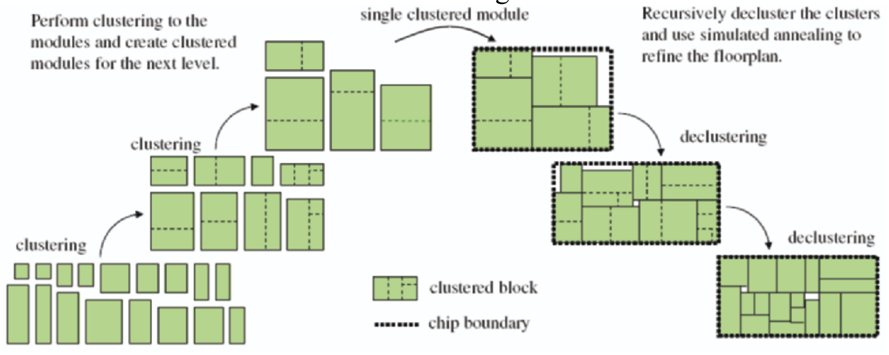

Perform clustering to the modules and create clustered modules for the next level.
single clustered module
Recursively decluster the clusters and use simulated annealing to refine the floorplan.
clustering
clustering
declustering
declustering
clustered block
chip boundary

Figure 9. The Λ-shaped multilevel framework of MB*-tree

The clustering metric is defined by the two criteria: area utilization (*dead space*) and the *connectivity density* among blocks.

- Dead space: The area utilization for clustering two blocks m_i and m_j can be measured by the resulting dead space s_{ij}, representing the unused area after clustering m_i and m_j. Let s_{tot} denote the dead space in the final floorplan P. We have $s_{tot} = A_{tot} - \sum_{m_i \in M} A_i$, where A_i denotes the area of block m_i and A_{tot} the area of the final enclosing rectangle of P. Since $\sum_{m_i \in M} A_i$ is a constant, minimizing A_{tot} is equivalent to minimizing the dead space s_{tot}. For the example shown in Figure 10, $s_{12} = 0$, $s_{13} = 36$, and $s_{tot} = 36$.
- Connectivity density: Let the connectivity c_{ij} denote the number of nets between two (primitive or cluster) blocks m_i and m_j. The *connectivity density* d_{ij} between two blocks m_i and m_j is given by

$$d_{ij} = c_{ij}/(n_i + n_j), \tag{4}$$

where n_i (n_j) denotes the number of primitive blocks in m_i (m_j). Often a bigger cluster implies a larger number of connections. The connectivity density considers not only the connectivity but also the sizes of clusters between two blocks to avoid possible biases. For the example shown in Figure 11, we apply the clustering scheme $\{\{m_1, m_2\}, \{m_3, m_4\}\}$ (based on connectivity density), instead of $\{\{\{m_1, m_2\}, m_3\}, m_4\}$ (based on connectivity).

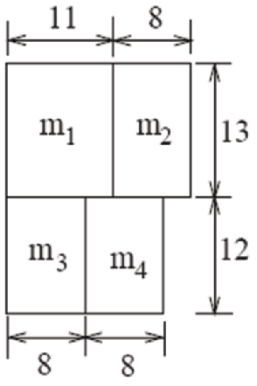


Figure 10. A cluster with the four primitive blocks, a, b, c, and d. The placement can be obtained by applying the clustering scheme $\{\{m_1, m_2\}, \{m_3, m_4\}\}$, resulting in a dead space of 36 units

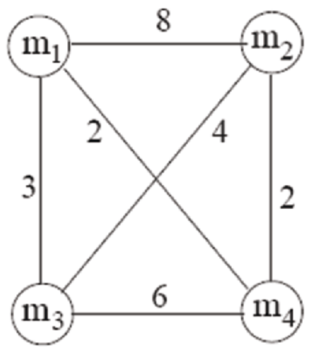


Figure 11. An example connectivity between each pair of blocks. We apply the clustering scheme $\{\{m_1, m_2\}, \{m_3, m_4\}\}$ based on connectivity density, instead of $\{\{\{m_1, m_2\}, m_3\}, m_4\}$ (based on connectivity)

Obviously, the cost function of dead space is for area optimization while that of connectivity density is for timing and wiring area optimization. Therefore, the metric for clustering two (primitive or cluster) blocks m_i and m_j, $\phi : \{m_i, m_j\} \to \Re^+ \cup \{0\}$, is then given by

$$\phi(\{m_i, m_j\}) = \alpha \hat{s}_{ij} + \frac{\beta K}{\hat{d}_{ij}}, \tag{5}$$

where $\hat{s}_{ij}$ and $K/\hat{d}_{ij}$ are respective normalized costs for s_{ij} and K/d_{ij}, α, β and K are user-specified parameters/constants. We set $K = \sum s_{ij} / \sum d_{ij}$ to normalize the dead space and the connectivity cost, i.e., to make the ranges of the two normalized costs about the same. Note that we shall normalize the dead space and connectivity density to equally weigh the two costs. To calculate the normalization factors for $s_{i,j}$ and $d_{i,j}$, we can preprocess using simulated annealing to derive the initial temperature and then obtain the approximate ranges of the resulting area and connectivity density to normalize the costs. For example, we may perform 100 runs of simulated annealing to obtain the approximate ranges of the resulting costs (i.e., area and connectivity density here) and derive the factors (weights) to equally weigh the costs by making the ranges of the two costs about the same. By doing so, it is more meaningful to weigh the area and connectivity density costs through the controlling factors α and β.

The declustering stage iteratively ungroups a set of previously clustered blocks (i.e., expanding a node into a subtree according to the B*-tree topology constructed at the clustering stage) and then refines the floorplan solution based on a simulated annealing scheme. The refinement shall lead to a "better" B*-tree structure that guides the declustering at the next level. It is important to note that we always keep only one B*-tree for processing at each iteration, and the agglomeratively multilevel B*-tree based floorlanner preserves the geometric relations among blocks during declustering (i.e., the tree expansion), which makes the B*-tree an ideal data structure for the multilevel floorplanning framework.

The declustering metric is defined by the two creiteria: area utilization (*dead space*) and the *wirelength* among blocks.

- Dead space: Same as that defined in the clustering stage.
- Wire length: The wirelength of a net is measured by half the bounding box of all the pins of the net, or by the length of the center-to-center interconnections between the blocks if no pin positions are specified. The wirelength for clustering two blocks m_i and m_j, w_{ij}, is measured by the total wirelength interconnecting the two blocks. The total wirelength in the final floorplan P, w_{tot}, is the summation of the length of the wires interconnecting all blocks.

Obviously, the cost function of dead space is for area optimization while that of wirelength is for timing and wiring area optimization. Therefore, the metric for refining a floorplan solution during declustering, $\psi_{tot} : M \rightarrow \Re^{+} \cup \{0\}$, is then given by

$$\psi_{tot} = \gamma \hat{s}_{tot} + \delta \hat{w}_{tot}, \tag{6}$$

where $\hat{s}_{tot}$ and $\hat{w}_{tot}$ are respective normalized costs for s_{tot} and w_{tot}, and γ and δ are user-specified parameters. Note that the normalization procedure for s_{tot} and w_{tot} is similar to that used for clustering.

MB*-tree scales very well as the circuit size increases. The capability of the MB*-tree shows its promise in handling large-scale designs with complex constraints.

2.4.3 The V-Shaped multilevel floorplanning

We describe the V-shaped multilevel floorplanning of top-down partitioning followed by bottom-up merging. Figure 12 illustrates a V-shaped interconnect-driven multilevel floorplanning framework (IMF for short) proposed by Chen, Chang, and Lin in [21].

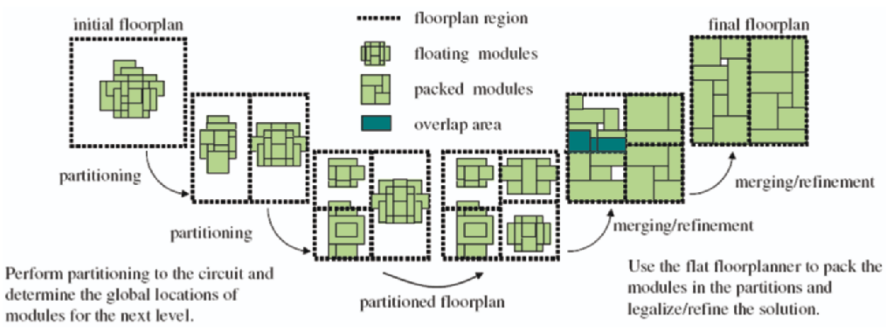

initial floorplan
floorplan region
floating modules
packed modules
overlap area
final floorplan
partitioning
partitioning
merging/refinement
merging/refinement
Perform partitioning to the circuit and determine the global locations of modules for the next level.
partitioned floorplan
Use the flat floorplanner to pack the modules in the partitions and legalize/refine the solution.

Figure 12. The V-shaped multilevel framework of IMF

At the initial level, the locations of all blocks are set to the center of the chip region. To prevent from generating sub-regions of large aspect ratios, we choose the longer side to divide the region into two sub-regions. After the shapes of two sub-regions are determined, we move the blocks to the two centers of the two sub-regions to minimize the half-perimeter wirelength (HPWL).

The block-location determination problem can be formulated as a hypergraph partitioning problem. We first derive an exact net-weight modeling to map the HPWL cost exactly to the min-cut cost. With the exact modeling, in other words, minimizing HPWL is equivalent to finding the min-cut cost. Therefore, the given hypergraph is partitioned using a min-cut bipartitioner to obtain the minimum HPWL. The new locations of the blocks are thus determined by the partitioner, and each sub-partition corresponds to a sub-region.

The partitioning stage continues until the number of blocks in each partition is smaller than a threshold. Then, the partitioned floorplan is obtained.

In the merging stage, we first use fixed-outline floorplanning to pack the blocks in the partition, and then merge two neighboring regions into one larger region. The fixed-outline floorplanning is applied again to legalize/refine the floorplan.

Each region has its own height and width, and all blocks in the region must fit into the region to generate a feasible floorplan. We treat the blocks and I/O pads outside the current region as fixed terminals. Fixed-outline floorplanning is applied to every region. Simulated annealing is used to find a feasible floorplan to fit all blocks into the region and minimize the wirelength. Then, two neighboring regions are merged into one larger region, and fixed-outline floorplanning is used again to refine the floorplan.

The cost function Φ can be defined as follows:

$$\Phi = k_1 \frac{A_F}{A_{F,norm}} + k_2 \frac{W_L}{W_{L,norm}} + k_3 \left(\frac{W_F}{H_F} - \frac{W_R}{H_R} \right)^2, \qquad (7)$$

where A_F is the current floorplan area, $A_{F,norm}$ is the area normalization factor, W_L is the current wirelength, $W_{L,norm}$ is the wirelength normalization factor, W_F is the current floorplan width, H_F is the current floorplan height, W_R is the width of the region, H_R is the height of the region, and k_1, k_2, k_3 are user-specified parameters. To calculate the area/wirelength normalization factors, several times of random perturbations are performed before simulated annealing starts, and $A_{F,norm}$ ($W_{L,norm}$) is set to the average value of A_F (W_L).

If the fixed-outline floorplanning cannot find a feasible floorplan within the bounding box, we still keep the solution. In the next refinement level, two partitioned regions are merged. To merge two vertical regions, we make the root of the B*-tree for the upper sub-floorplan as the right child of the right-most node of the B*-tree for the bottom sub-floorplan. Figure 13 shows an example of vertical merging. After merging, the root n_0 of T_1 becomes the right child of the right-most node n_6 of T_2. The width of the merged floorplan is equal to the maximum width of the sub-floorplans, and the height of the floorplan is less than or equal to the sum of the two sub-floorplan's heights due to the packing. To merge two horizontal regions, we first find the node which corresponds to the right-most block of the left sub-floorplan. Then, we make the root of the B*-tree for the right sub-floorplan as the left child of the node we found. Figure 14 shows an example of horizontal merging. The node n_3 corresponds to the right-most block of the left sub-floorplan. The root n_4 of T_2 becomes the left child of the node n_3 of T_1. The height of the merged floorplan is equal to the maximum height of the two sub-floorplans, and the width of the merged floorplan is equal to the sum of the two sub-floorplan's widths.

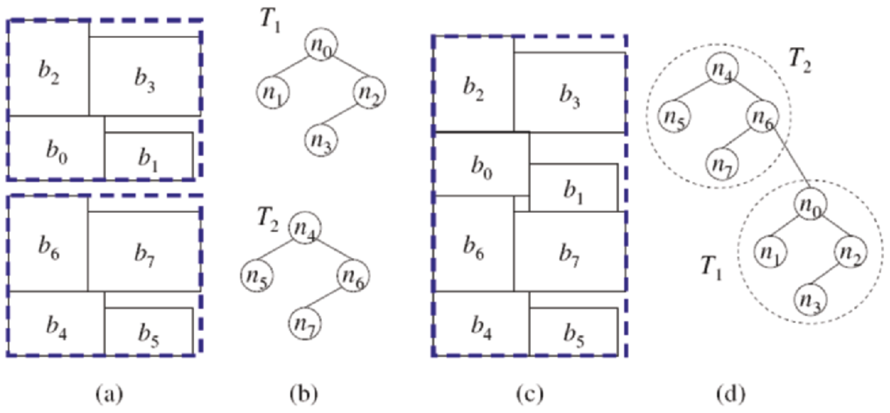


Figure 13. An example of vertical merging. (a) Two sub-floorplans. (b) The corresponding B*-trees. (c) The merged floorplan. (d) The merged B*-tree

The merging stage iteratively merges two previously partitioned regions and then refines the floorplan solution based on fixed-outline simulated annealing. The merging stage continues until all regions are merged into one top-most region, and the final floorplan is obtained.

2.4.4 Framework comparison

In addition to the multilevel frameworks, hierarchical approaches are also proposed to cope with the scalability problem. The hierarchical approaches recursively divide a floorplanning region into a set of sub-regions and solve those sub-problems independently. Adya et al. [2] propose a "floorplacement" framework (used in their program Capo 9) that combines partitioning and floorplanning techniques to handle both floorplanning and placement problems. It first partitions a floorplan and then finds legal sub-floorplans. Cong et al. [34] present a fast floorplanner called PATOMA using look-ahead enabled recursive bipartitioning. It partitions a floorplan and uses row-oriented block (ROB) packing and zero-dead space (ZDS) floorplanning to find legal sub-floorplans. Both the floorplacement and PATOMA are based on the hierarchical framework in which the floorplanning stage is only used for legalization and overlap removal. The top-down hierarchical technique is efficient in handling large-scale problems. Nevertheless, a drawback of the hierarchical approaches is that they might lack the global information for the floorplanning interaction among different sub-regions, for which the multilevel frameworks are proposed to remedy the deficiency.

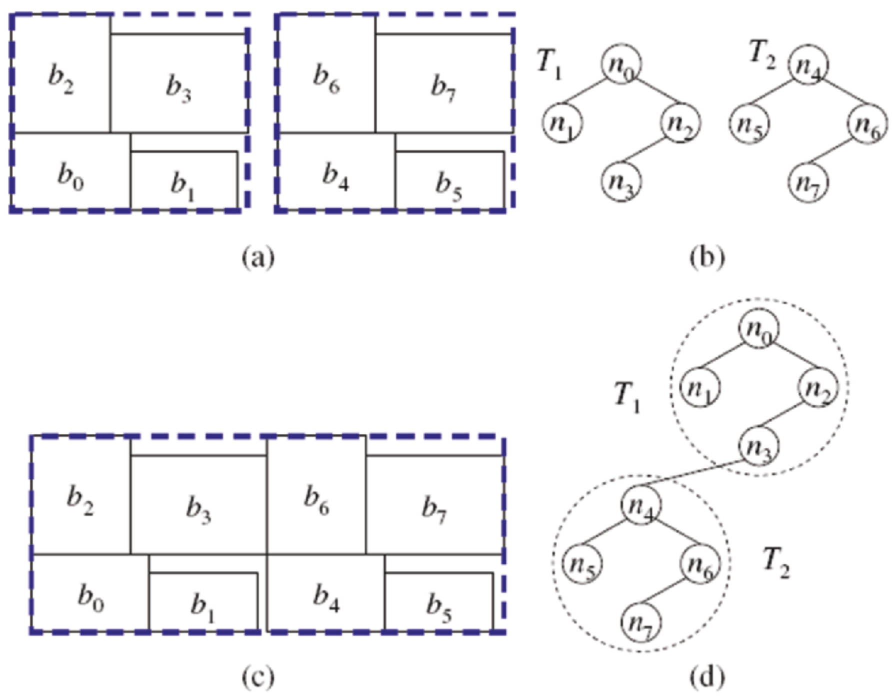


Figure 14. An example of horizontal merging. (a) Two sub-floorplans. (b) The corresponding B*-trees. (c) The merged floorplan. (d) The merged B*-tree

Table 1 lists the characteristics of the Capo floorplacement [2] and the PATOMA frameworks [34], the MB*-tree multilevel framework [76], and the IMF multilevel framework [21]. The IMF framework and the MB*-tree framework are based on the multilevel framework while the Capo floorplacement and the PATOMA frameworks are based on the hierarchical framework. Although Capo and PATOMA use partitioning, unlike IMF, they do not have the refinement stage to further improve their results.

Table 1. Framework comparisons.

Framework	Characteristics
The Capo floorplacement framework in [2]	• Use the top-down hierarchical framework. • Use partitioning and fixed-outline floorplanning. • Minimize the wirelength under the given chip-outline. • Do not have a refinement stage.
The PATOMA framework in [34]	• Use the top-down hierarchical framework. • Use partitioning and ZDS/ROB fast look-ahead floorplanning. • Minimize the wirelength under the given chip-outline. • Do not have a refinement stage.
The MB*-tree multilevel framework in [76]	• Use the Λ -shaped multilevel framework.. • Use bottom-up clustering followed by top-down declustering. • Deal with variable dies and cannot guarantee to satisfy an outline constraint. • Need to specify the weights for area and wirelength by the user.
The IMF multilevel framework in [21]	• Use the V-shaped multilevel framework. • Use top-down partitioning followed by bottom-up merging. • Handle fixed-die constraints. • Minimize the wirelength under the given area constraint.

2.5 Floorplanning Considering Substrate Noise

More and more SOC designs require the integration of analog and digital circuits on a single chip and would therefore suffer from substrate noise coupling. With the growing of system frequency, some existing techniques designed for reducing substrate noise may not work well. Considering substrate noise in early floorplanning is thus desirable. We introduce in this section a pioneering work along this direction by Cho, Shin, and Pan [27].

For efficient simulation of large SOCs, a simple model that accurately predicts substrate coupling must be used. The substrate coupling model used in [27] is scalable with contact shapes, dimensions, and separations. It also considers the issues related to package parasitic, backplane connections, and noise suppression techniques. The substrate is modelled by a two-port lumped resistor network as shown in Figure 15, but it is only valid for frequency below a few gigahertz. The resistance values are determined by characterizing the substrate either through device simulations or through measurements of the substrate. The scalable macro-model is based on Z-parameters from the derived resistances as follows:

$$Z = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix} = \frac{1}{\Delta}\begin{bmatrix} G_D + G_{DA} & G_{DA} \\ G_{DA} & G_A + G_{DA} \end{bmatrix}, \tag{8}$$

where $\square = G_A G_{DA} + G_D G_{DA} + G_A G_D$. $Z_{11}(Z_{22})$ can be expressed as [90]

$$Z_{11} = \frac{1}{K_1 Area + K_2 Perimeter + K_3}, \tag{9}$$

where K_1, K_2 *and* K_3 are process parameters. Z_{12} is given by

$$Z_{12} = \alpha e^{-\beta x}, \tag{10}$$

where x is the distance between contacts and α and β are process parameters. The substrate coupling can be calculated from the value of resistors in the two-port lumped network shown in Figure 15 The substrate coupling if i-th digital block to j-th analog block, SC_{ij} is given by

$$SC_{ij} = \frac{R_A}{R_A + R_{DA}} = \frac{G_{DA}}{G_{DA} + G_A} = \frac{Z_{12}}{Z_{22}}. \tag{11}$$

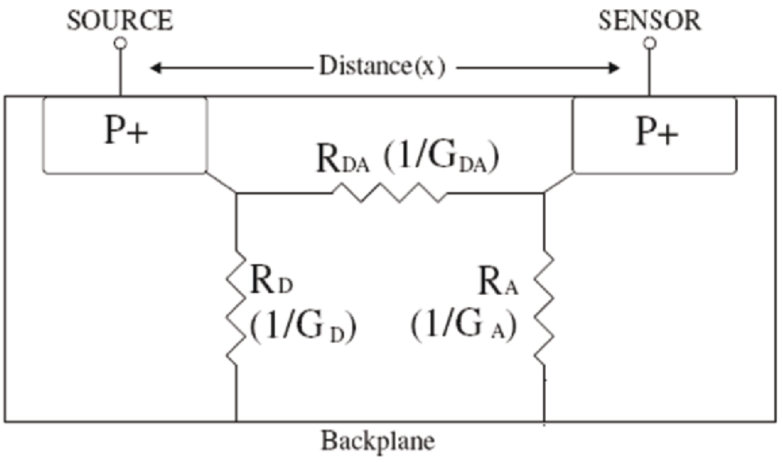


Figure 15. Macro-model for the substrate

Because of the frequency-dependent characteristics of noise source and sensor block, we should consider frequency when calculating substrate noise. Based on a functional analog block description, it is possible to determine, with reasonable accuracy, the type of frequently used analog blocks. We consider only substrate noise due to power/ground (P/G) bounces, and use the P/G bounce limits as a means of early substrate noise characterization of a digital block.

The substrate noise of j-th analog block from the switching of i-th digital block, $N_{i,j}$, can be approximated by [10]

$$N_{i,j} = \sqrt{\sum_j (SC_{i,j})^2 \cdot \int_{freq\_low}^{freq\_high} (S_i(f) \cdot N_j(f))^2 \, df}, \tag{12}$$

where $S_i(f)$ ($N_j(f)$) can be derived from typical sensitivity characteristics of analog (digital) blocks. The total noise from all digital blocks is

$$N_{total} = \sum_i \sum_j N_{i,j}. \tag{13}$$

In the substrate noise model, some important variables determine substrate noise. These variables contain process parameters, areas and perimeters of blocks, distance between blocks and frequency-dependent characteristics of the noise source and sensor block. In the early floorplanning, we may minimize substrate noise by changing the distance between blocks and the perimeter of a block (soft block) and consider characteristics of a block at the same time.

2.5.1 Block preference directed graph

The substrate noise model described above is one of the most compact models with high scalability and accuracy, but it is still computationally expensive to perform a substrate noise estimation. For fast substrate noise estimation, we can construct a block preference directed graph (BPDG for short) [27]. The BPDG construction consists of substrate noise table construction, analog block ordering, digital block ordering, and BPDG construction.

To eliminate the effect of distance, we assume that the nominal distance is fixed. With fixed distance, the substrate noise between a digital and an analog block purely depends on frequency coupling and geometric properties like area and shape. For example, Table 2 quantifies the substrate noise with fixed distance.

Table 2. Substrate noise table

	D_1	D_2	D_3	D_4	D_5	D_6
A_1	5	2	6	3	10	1
A_2	2	1	3	10	8	5
A_3	3	8	7	11	9	12

Based on the substrate noise table, analog blocks can be sorted for each digital block by descending order of substrate noise. Figure 16 gives an example based on Table 2.

$$\boxed{\begin{array}{l} D_1 \ : \ A_1 \leftarrow A_3 \leftarrow A_2 \\ D_1 \ : \ A_3 \leftarrow A_1 \leftarrow A_2 \\ D_1 \ : \ A_3 \leftarrow A_1 \leftarrow A_2 \\ D_1 \ : \ A_3 \leftarrow A_2 \leftarrow A_1 \\ D_1 \ : \ A_1 \leftarrow A_3 \leftarrow A_2 \\ D_1 \ : \ A_3 \leftarrow A_2 \leftarrow A_1 \end{array}}$$

Figure 16. Analog block orderings

Based on the substrate noise table, digital blocks can be sorted for each analog block by the ascending order of substrate noise.

$$\boxed{\begin{array}{l} A_1 \ : \ D_6 \leftarrow D_1 \leftarrow D_4 \leftarrow D_1 \leftarrow D_3 \leftarrow D_5 \\ A_1 \ : \ D_2 \leftarrow D_2 \leftarrow D_3 \leftarrow D_6 \leftarrow D_5 \leftarrow D_4 \\ A_1 \ : \ D_1 \leftarrow D_3 \leftarrow D_2 \leftarrow D_5 \leftarrow D_4 \leftarrow D_6 \end{array}}$$

Figure 17. Analog block orderings

Then, we can construct BPDG (see Figure 18) by the BPDG_Construction algorithm listed in Figure 19. Lines 1 and 2 initialize

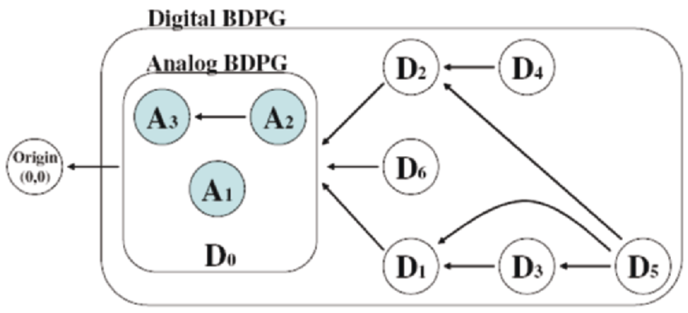


Figure 18. Block preference directed graph

G_a and G_d to be empty. Lines 3–7 check if A_i is before A_j in all O_a and add a directed edge from A_j to A_i in G_a if it is true. It means that if there is an edge from A_j to A_i, A_i should be farther than A_j to all digital blocks. Lines 8–12 check if D_i is before D_j in all O_d and add a directed edge from D_j to D_i in G_d if it is true. If there is an edge from D_j to D_i, D_i should be closer than D_j to all analog blocks. Line 13 add a virtual vertex D_0 that contains G_a. Line 14 connects the relation between analog and digital blocks.

```
Algorithm: BPDG_Construction
Input: Analog, digital block orderings $O_a$ and $O_d$
1  Analog BPDG $G_a \leftarrow \emptyset$
2  Digital BPDG $G_d \leftarrow \emptyset$
3  for each analog block $A_i$ and $A_j$, $i \neq j$ do
4      if $A_i$ is before $A_j$ in all $O_a$ then
5          Add a directed edge from $A_j$ to $A_i$ to $G_a$
6      endif
7  endfor
8  for each digital block $D_i$ and $D_j$, $i \neq j$ do
9      if $D_i$ is before $D_j$ in all $O_d$ bthen
10         Add a directed edge from $D_j$ to $D_i$ to $G_d$
11     end if
12 end for
13 Add a virtual vertex $D_0$ for $G_a$ to $G_d$
14 Add directed edges from all root vertices to $D_0$
Output: $G_d$.
```

Figure 19. The BPDG construction algorithm

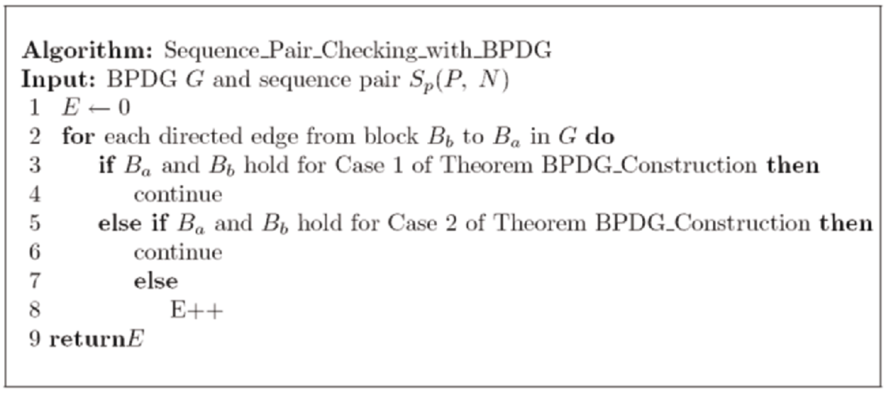
2.5.2 Sequence pair with BPDG

The substrate noise aware floorplanning presented in [27] is based on the sequence pair floorplan representation. According to the properties of sequence pair, we have the following theorem.

Theorem 1 *A block B_a is guaranteed to have shorter distance to the left-bottom corner than a block B_b under a completely packed floorplan, if either of the following conditions is satisfied.*

1. *There is no block* B_s *satisfying* $LCS(X_1, Y_1) = \varnothing$ *in a sequence pair* $(P, N) = (\ldots B_a X_1 B_s \ldots B_b, \ldots B_s Y_1 B_a \ldots B_b \ldots)$.
2. *There is no block* B_s *satisfying* $LCS(X_2, Y_2) = \varnothing$ *in a sequence pair* $(P, N) =$ $(\ldots B_b \ldots B_s X_2 B_a \ldots, \ldots B_s Y_2 B_a \ldots B_b \ldots)$.

In the Sequence_Pair_Checking_with_BPDG algorithm listed in Figure 20, line 1 initializes the violation number to zero. Lines 2–10 check that, for each directed edge from block B_b to B_a in G, the violation number would be increased if B_a and B_b do not hold for any of the cases in Theorem 1. To speed up the process, the checking is performed incrementally. If the movement which may happen in X_1, X_2, Y_1, and Y_2 of Theorem 1, the checking result would be the same as the previous one.



```
Algorithm: Sequence_Pair_Checking_with_BPDG
Input: BPDG $G$ and sequence pair $S_p(P, N)$
1 $E \leftarrow 0$
2 for each directed edge from block $B_b$ to $B_a$ in $G$ do
3     if $B_a$ and $B_b$ hold for Case 1 of Theorem BPDG_Construction then
4         continue
5     else if $B_a$ and $B_b$ hold for Case 2 of Theorem BPDG_Construction then
6         continue
7     else
8         E++
9 return$E$
```

Figure 20. Sequence Pair Checking with BPDG

2.5.3 Fast substrate noise-aware floorplanning

Figure 21 shows the fast substrate noise-aware floorplanning algorithm. Line 1 performs the floorplanning with analog blocks first to bind all the analog blocks together. Line 2 allocates a guard ring around the analog block cluster to mitigate the substrate noise. Line 3 takes all the analog blocks as a virtual block B_v. Line 4 performs the floorplanning for all digital blocks and B_v. The cost function is

$$Cost = \alpha \frac{F}{F_r} + \beta \frac{NV}{NV_r}, \tag{14}$$

```
Algorithm: Fast Substrate Noise-Aware Floorplanning
Input: Analog BPDG $A_g$, Digital BPDG $D_g$
1 Perform floorplanning with analog blocks with $A_g$;
2 Inflate the analog block floorplan;
3 Make the analog floorplan as a virtual block $B_v$;
4 Perform floorplanning with digital blocks and $B_v$ with $D_g$;
5 returnThe final floorplan.
```

Figure 21. Fast Substrate Noise-Aware Floorplanning

where F is the cost of conventional digital floorplanning, NV is the number of violations. Fr and NVr are reference values for each cost factor. α and β are coefficients for balancing two cost factors. NV is returned by the Sequence Pair checking with BPDG algorithm shown in Figure 20 after a perturbation.

2.6 Bus-Driven Floorplanning

An SOC design needs to integrate various IP blocks, and the communication among those blocks are often conducted on system buses. Floorplanning with bus planning is one of the most challenging modern floorplanning problems because it needs to consider the constraints with interconnect and block positions simultaneously. In this section, we introduce the B*-tree based bus-driven floorplanning algorithm presented in [22].

2.6.1 Bus-driven floorplanning formulation

We consider a chip with multiple metal layers, and buses are assigned on the top two layers. The orientation of buses is either horizontal or vertical. The problem of bus-driven floorplanning (BDF) is defined as follows [106]:

Given n rectangular macro blocks $B = \{b_i \mid i = 1,...,n\}$ and m buses $U = \{u_i \mid i = 1,...,m\}$, each bus u_i has a width t_i and goes through a set of blocks B_i, where $B_i \subseteq B$ and $|B_i| = k_i$. Decide the positions of macro blocks and buses such that there is no overlap between any two blocks or between any two horizontal (vertical) buses, and bus u_i goes through all of its k_i blocks. At the same time, the chip area and the bus area are minimized.

For convenience, let $< g, t, \{b_1, ..., b_k\} >$ represent a bus u where $g \in \{H, V\}$ is the orientation, t is the bus width, and $b_i, i = 1, ..., k$, are the blocks that the bus goes through. For short, a bus is represented by $\{b_1, ..., b_k\}$. Figure 22 shows a feasible horizontal bus.

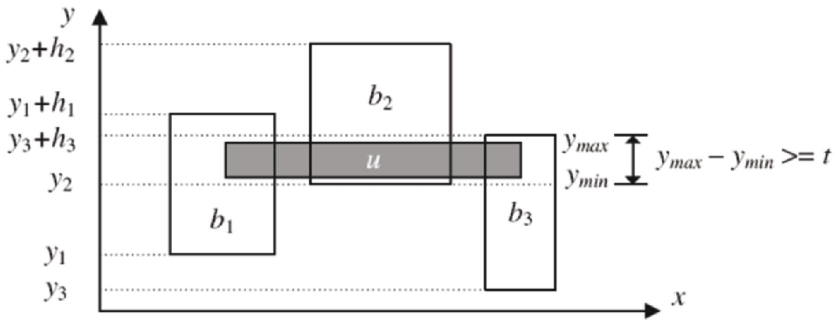


Figure 22. A feasible horizontal bus $u =< H, t, \{b_1, b_2, b_3\} >$

2.6.2 B*-tree properties for bus constraints

The blocks that a bus goes through must locate in an alignment range, i.e., the vertical or horizontal overlap of the blocks has to be larger than the bus width. For a B*-tree, the left child n_j of the node n_i represents the lowest adjacent block b_j which is right to the block b_i (i.e. $x_j = x_i + w_i$). So, the blocks have horizontal relationships in a left-skewed sub-tree.

Property *1 In a B*-tree, the nodes in a left-skewed sub-tree may satisfy a horizontal bus constraint.*

Blocks are compacted to the bottom and left after packing. So the blocks associated with a left-skewed sub-tree of a B*-tree may be aligned together if no block falls down during packing. We introduce *dummy blocks* to solve the falling down problem. In Figure 23(a), the blocks b_2 and b_4 are displaced because they fall down during packing. We add dummy blocks right below the displaced blocks. The dummy blocks have the same x-coordinates as the displaced blocks, and the widths are also the same. In Figure 23(b), we adjust the heights of dummy blocks to shift the displaced blocks to satisfy the bus constraint. After adjusting the heights of dummy blocks, we can guarantee that the blocks are feasible with the horizontal bus

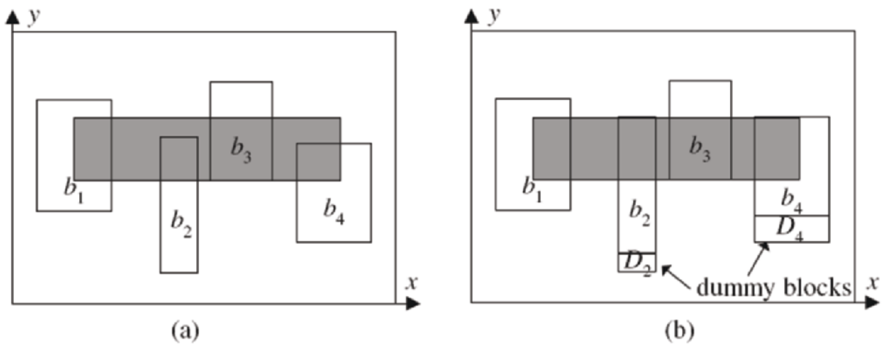


Figure 23. (a) An infeasible floorplan since the block-overlap range is less than the bus width t. (b) Inserting dummy blocks, the bus $<H, t, \{b_1, b_2, b_3, b_4\}>$ is satisfied

constraint. The height Δ_i of the dummy block D_i can be computed by the following equation:

$$\Delta_i = \begin{cases} (y_{min} + t) - (y_i + h_i), & \text{if } (y_{min} + t) > (y_i + h_i) \\ 0, & \text{otherwise,} \end{cases} \tag{15}$$

where x_i (y_i) is the x (y)-coordinate of block b_i, and $y_{min} = \max\{y_i \mid i = 1, 2, ..., k\}$ for a bus $\{b_1, ..., b_k\}$. Figure 24 shows an example of a feasible horizontal bus by inserting dummy blocks D_5 and D_6.

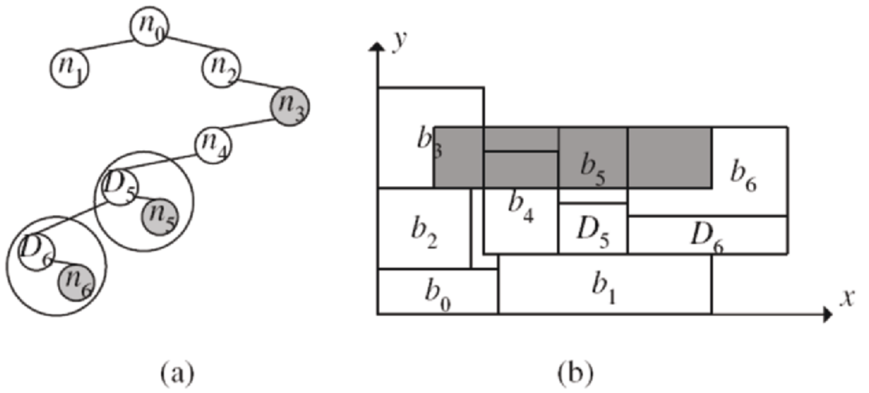


Figure 24. (a) The B*-tree with a left-skewed sub-tree after inserting dummy nodes. (b) The corresponding feasible horizontal bus $<H, t, \{b_3, b_5, b_6\}>$

Property 2 *By inserting dummy blocks of appropriate heights, we can guarantee the feasibility of a horizontal bus with blocks whose corresponding B*-tree nodes are in a left-skewed sub-tree.*

For a B*-tree, the right child n_j of the node n_i represents the closest upper block b_j which has the same x-coordinate as the block b_i (i.e $x_j = x_i$). Therefore, the blocks in the right-skewed sub-tree are aligned with the x-coordinate. Assume the minimum width of the macro blocks that the bus goes through is larger than the bus width. The structure forms a vertical bus. In the example shown in Figure 25, the nodes n_3 and n_5 is in the right-skewed sub-tree of n_0, so the blocks b_0, b_3, and b_5 satisfy the vertical bus constraint.

Property *3 In a B*-tree, the nodes in a right-skewed sub-tree can guarantee the feasibility of a vertical bus.*

Note that the vertical bus is not constrained to be at the right sub-tree of the node corresponding to the lowest block among the set. According to Property 3, the nodes in a right-skewed sub-tree can guarantee the feasibility of a vertical bus, but it is not always true vice versa.

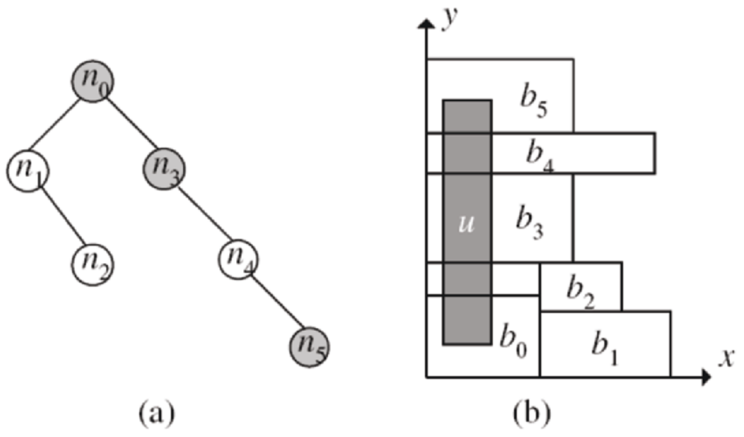


Figure 25. (a) A right-skewed sub-tree. (b) The corresponding feasible vertical bus $u = < V, t, \{b_0, b_4, b_5\} >$

2.6.3 The twisted-bus structure

Consider two buses simultaneously, we cannot always fix the horizontal bus constraints by inserting dummy blocks. As the example shown in Figure 26, two buses are considered: $u = \{b_0, b_3\}$ and $v = \{b_2, b_6\}$. We can add the dummy block D_0 (D_2) below b_0 (b_2) to satisfy the horizontal bus u (v). However, we cannot satisfy two horizontal bus constraints at the same time since two buses are twisted. The idea to discard B*-trees with twisted-bus structures is to reduce the solution space and make the solution searching more efficient. Note that it is impossible to fix a twisted-bus structure by inserting dummy blocks. Discarding such a configuration will not remove any feasible solutions. We directly examine the twisted-bus structure by checking the B*-tree nodes. Consider two buses u and v. If one node of bus u is in the right-skewed sub-tree of one node of bus v, and one node of bus v is in the right-skewed sub-tree of one node of bus u, then it will incur a twisted-bus structure. Therefore, we shall discard a B*-tree with such an infeasible tree topology during solution perturbation. Note that not all potential twisted-bus structures are checked through the aforementioned procedure. Figure 26 shows a twisted-bus structure where n_3 is in the right-skewed sub-tree of n_2, and n_6 is in the right-skewed sub-tree of n_0.

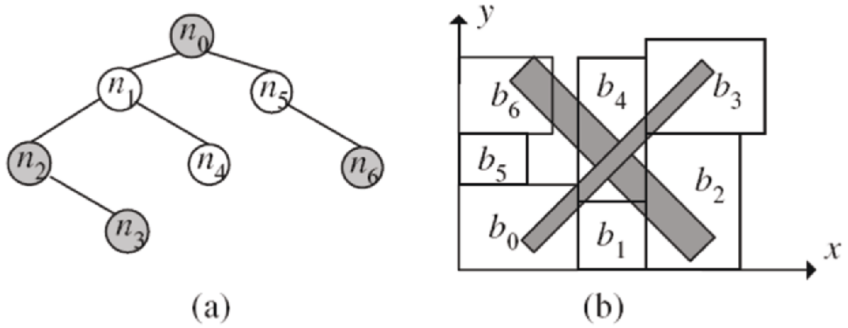


Figure 26. An infeasible floorplan for two buses, $u = \{b_0, b_3\}$ and $v = \{b_2, b_6\}$. (a) A twisted-bus structure where n_3 is in the right sub-tree of n_2, and n_6 is in the right sub-tree of n_0. (b) The corresponding floorplan. The two twisted-bus cannot be satisfied simultaneously by inserting dummy blocks

2.6.4 Bus overlapping

When multiple buses are considered, we need to avoid overlaps between buses. For example, in Figure 27, two horizontal buses are to be assigned. The buses $u = \{b_0, b_4\}$ ($v = \{b_2, b_3\}$) are feasible when we consider only one bus. However, the vertical space is not large enough for fitting two buses. In this case, we compute the minimum shifting distance for the block b_2, and

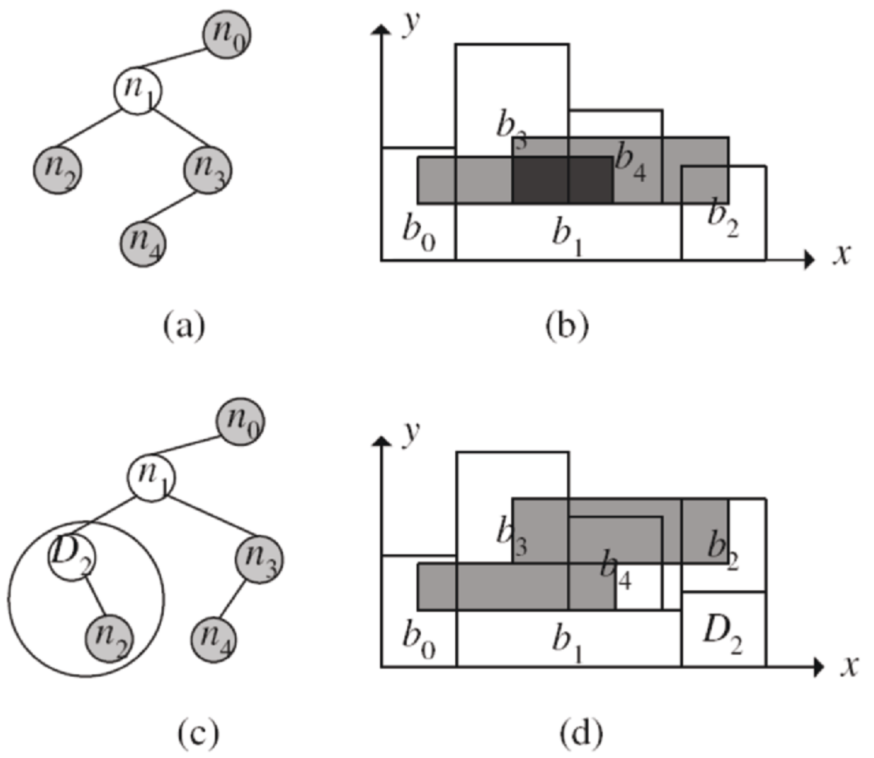


Figure 27. Two horizontal buses, $u = \{b_0, b_4\}$ and $v = \{b_2, b_3\}$. (a) Two buses overlap. (b) By inserting a dummy block, we can get a feasible floorplan without bus-overlapping

insert a dummy block D_2 right below b_2. Thus, the two buses can be assigned at the same time by inserting the dummy block. In their implementation, they check the buses one by one using the order in the benchmark. When one bus is examined, we also allocate the space for the bus according to the bus width and the block positions. If we find the space of one bus overlapping with another bus, we will let the new bus be on top of the other and insert dummy blocks to avoid overlaps.

2.6.5 Fixed I/O ports

Sometimes buses are connected to I/O ports that are fixed on the boundary. If the I/O ports to which the bus connects are at the top/bottom side of the chip, only the vertical bus may be feasible. Similarly, if the I/O ports to which the bus connected are at the left/right side of the chip, only the horizontal bus may be feasible. Thus, the directions of buses can be fixed, and we do not need to check the directions when deciding the bus locations.

To avoid the block falling down problem with fixed I/O ports, we need to set the heights of dummy blocks considering the positions of fixed I/O ports. We can directly set y_{min} in Equation (18) to the y-coordinate value of the fixed I/O port to which the bus connects. By doing so, it will try to align blocks with the fixed I/O port to make the horizontal bus feasible.

Figure 28 shows an example of inserting dummy blocks, considering a fixed I/O port F. The heights for dummy blocks D_3, D_5, and D_6 are $(y_F + t) - (y_3 + h_3)$, $(y_F + t) - (y_5 + h_5)$, and $(y_F + t) - (y_6 + h_6)$, respectively, where y_i is the y-coordinate of block i, and t is the bus width. The bus $\{b_3, b_5, b_6, F\}$ is feasible after inserting dummy blocks.

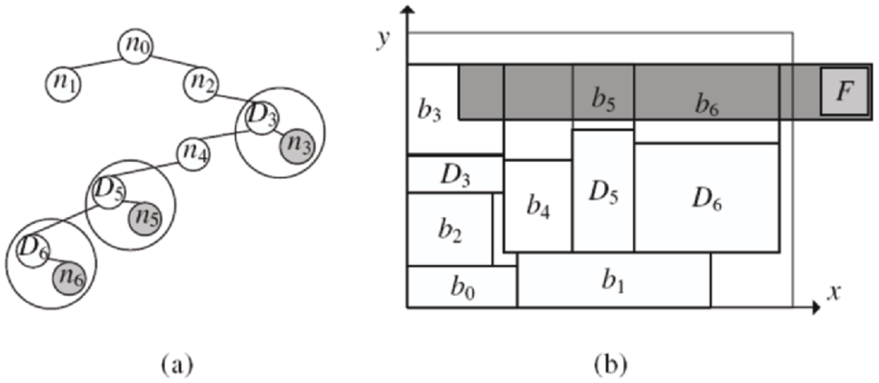


Figure 28. A horizontal bus connects to a fixed I/O port F, $u = \{b_3, b_5, b_6, F\}$. (a) The B*-tree after inserting dummy nodes. (b) The corresponding feasible horizontal bus

2.6.6 Algorithm

The bus-driven floorplanning algorithm applies simulated annealing based on the B*-tree representation. Figure 29 summarizes the algorithm. First, we initialize the B*-tree as a complete binary tree and start with the Fast-SA process. After each perturbation and non-dummy block packing, we check if there exists a “twisted-bus structure” in the B*-tree. If any, we simply discard the current solution and perturb the B*-tree again. This checking can save time to find feasible solutions. If there is no twisted-bus structure in the B*-tree, we insert the dummy blocks to the appropriate nodes to fix the horizontal bus constraints and bus-overlapping. After adjusting the heights of dummy blocks, we pack the B*-tree again. Then, we decide the bus locations so that there is no overlap between buses. After adjusting the heights of dummy blocks and re-packing the floorplan, some buses still may not be feasible. We refer to these buses as *unassigned buses*.

Since the objective function of bus-driven floorplanning is to satisfy all bus constraints so that the chip area and the total bus area are minimized, we define the cost function Ψ for a floorplan solution F with the set of buses U as follows:

$$\Psi(F,U) = \alpha A + \beta B + \gamma M, \qquad (16)$$

where A is the chip area, B is the bus area, M is the number of unassigned buses, and α, β, and γ are user-specified parameters.

In the SA process, we record the floorplan solution with the most number of feasible buses and the lowest cost. After the SA process stops, we report the lowest cost with the least number of unassigned buses. Thus, we can find the desired floorplan with the most feasible buses.

Suppose we are given m buses and n blocks. According to Figure 29, the combination of the pseudo code from line 4 to line 13 makes one

```
Algorithm: Bus-Driven Floorplanning Using B*-trees
Input: A set of blocks and a set of bus constraints.
Output: A floorplan satisfying bus constraints with
        chip area and the total bus area being minimized.
1   Initialize a B*-tree for the input blocks;
2   // Perform the simulated annealing process;
3   do
4       Perturb the B*-tree;
5       Pack macro blocks without dummy blocks;
6       if there exists a "twisted-bus structure" in the B*-tree
7           then restart the do-loop;
8       Adjust the heights of dummy blocks to fix horizontal
            bus constraints and fix bus-overlapping;
9       Pack macro blocks with dummy blocks;
10      Decide bus locations;
11      Evaluate the floorplan cost;
12      Decide if we should accept the new B*-tree.
13      Update the temperature;
14  until converged or cooling down;
15  return the best solution.
```

Figure 29. The bus-driven floorplanning algorithm

perturbation and evaluation of the B*-tree. Line 4 takes $O(1)$ time for perturbation, and Line 5 takes $O(n)$ time for B*-tree packing. In line 6, the time complexity for fixing horizontal bus constraints and bus overlap checking are $O(mn)$ and $O(m^2n)$, respectively. Line 9 takes $O(n)$ time for packing, Line 10 takes $O(m^2n)$ time for deciding the bus locations, and Lines 11–13 take $O(1)$ time. Thus, the total time complexity is $O(m^2n)$.

2.7 Conclusion

Floorplanning is an essential design step for hierarchical, building-block design methodology. It provides valuable insights into the hardware decision and estimation of various costs. The most popular floorplanning method resorts to the modelling of the floorplan structure and then optimizing the floorplan solutions using simulated annealing. There exist many floorplan representations in the literature. Yet, B*-tree and Sequence Pair have been recognized as the two most valuable representations due to their superior simplicity, effectiveness, efficiency, and flexibility.

To handle the challenges in modern SOC designs with large-scale circuit blocks, the multilevel floorplanning frameworks are desired. Two types of multilevel frameworks, the Λ- and V-shaped frameworks, have recently been studied in the literature. Both are based on two-stage techniques. The Λ-shaped framework adopts bottom-up coarsening followed by top-down uncoarsening, while the V-shaped framework proceeds with top-down uncoarsening and followed by bottom-up coarsening. Since the V-shaped framework processes global layout regions first, it tends to obtain better solutions for those with global effects such as wirelength and timing. In contrast, the Λ-shaped framework tends to achieve better solutions for local effects such as area optimization.

An SOC typically consists of various digital and analog functional blocks and interconnects them with system buses and/or global wiring. Therefore, it is crucial to consider the floorplanning with both digital and analog blocks and plan the system bus as early as possible. This section provides underlying ideas for handling the SOC floorplanning problems using the B*-tree and Sequence Pair formulations. Future research on SOC floorplanning lies in the considerations of various placement constraints (position constraints, thermal electrical constraints, etc.) and the co-synthesis of floorplan with other design targets (power/ground network, timing, noise, etc.).

3. PLACEMENT

3.1 Introduction

As the process technology advances, the feature size is getting smaller and smaller, which makes it possible to integrate an entire system with one billion transistors on a single chip. Two challenges arise due to this design complexity. First, the Intellectual Property (IP) blocks and pre-designed macro blocks (such as embedded memories, analog blocks, pre-designed datapaths, etc.) are often reused, and thus many IC's contain thousands of macro blocks and millions of cells. Second, timing optimization becomes more challenging due to the design complexity and the scaling of devices and interconnects.

The traditional placement problem seeks to minimize wirelength under the constraint that cells/macros do not overlap with each other. Three types of most popular techniques are used in the current state-of-the-art placers: (1) the partitioning based approach [2,25,70], (2) the simulated annealing based approach, and (3) the analytical approach [66,15]. Based on the techniques, many mixed-size placement algorithms are developed, which can be classified into three categories. The first category places macros and standard cells simultaneously, such as APlace [66], Feng Shui [70], mPG-MS [16], mPL [15], and UPlace [108]. The second category combines floorplanning and placement techniques, such as Capo [2]. The third category works in two stages: first place the macros and then the standard cells, such as the algorithm presented in [4].

In this chapter, we introduce wirelength- and timing-driven placement with various constraints for SOC designs, which usually has large-scale, mixed-size cells/blocks.

3.2 Problem Definition

We are given a set of m rectangular blocks (circuit blocks or cells) $B = \{b_1, b_2, ..., b_p\}$ whose width, height, and area are denoted by w_i, h_i, and a_i, $1 \le i \le p$, a netlist $N = \{n_1, n_2, \ldots, n_k\}$, and a set of locations (slots) $L = \{l_1, l_2, \ldots, l_q\}$, $p \le q$. The placement problem is to assign each $b_i \in B$ to a unique location $P_j = (x_i, y_i)$ on the chip layout such that no two blocks overlap with each other (i.e., legalization constraint) and some objective (such as the total wirelength, congestion, timing) is optimized. There exist a few popular estimations for the wirelength; for example, half bounding box of the interconnection (also known as the semi-perimeter method), minimum spanning tree approximation, squared Euclidean distance (squares of all pairwise terminal distances in a net using a quadratic cost function) [72], the log-sum-exp method [44], etc.

3.3 Approaches to Placement

Three types of most popular techniques are used in the state-of-the-art placers: (1) the partitioning based approach, (2) the simulated annealing based approach, and (3) the analytical approach. Independent of the placement techniques used, most modern placers consist of three major steps: (1) global placement, (2) cell legalization, and (3) detailed placement (see Figure 30). We detail these approaches in the following.

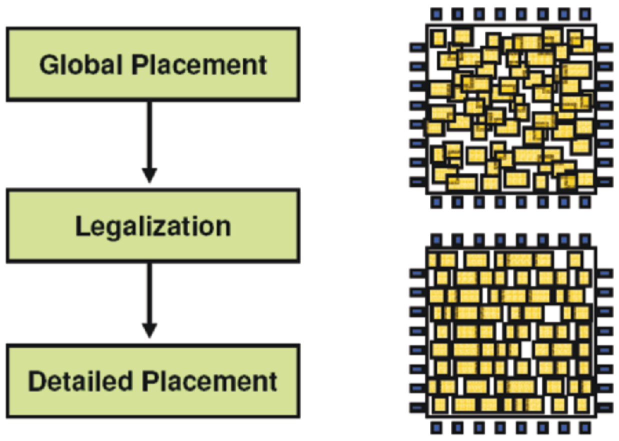


Figure 30. Three major steps in placement

3.3.1 Partitioning-based methods

Among academic placement tools, all the leading top-down methods rely on variants of recursive circuit partitioning in someway. An early work on partitioning-based placement was proposed by Dunlop and Kernighan [41]. Most modern methods, including Capo [13], Feng Shui [109], and NTUplace [25,61], have exploited further advances in fast algorithms for hypergraph partitioning to push these frameworks beyond their original capabilities. Fast, high-quality $O(n)$ partitioning algorithms give top-down partitioning attractive $O(n \lg n)$ scalability overall, where n is the problem size.

In the following, we introduce the underlying ideas of the three major steps of the NTUplace partitioning-based placer: (1) global placement, (2) cell legalization, and (3) detailed placement.

A. Global Placement

A.1. Cutline Determination

At each level of global placement, NTUplace determines the cutline position at each partitioning step for better bi-partitioning with more balanced cell density. The cutline position has a significant impact on the chip density and the accuracy of terminal propagation. To find a better cutline position and perform more accurate terminal propagation, we pre-partition a circuit with a relaxed balance factor. Then, it moves the cutline to make the free space ratio of the two subregions equal to the size ratio of the min-cut pre-partitioning result. The cutline is then used to guide the exact net-weight modeling with terminal propagation [21]. Finally, it reapplies the min-cut partitioning with a tighter balance factor to obtain the final partitioning solution.

A.2. Exact Net-Weight Partitioning

Simple recursive bisection with a cutsize objective can be used quite effectively with simple Fiduccia-Matheysses (FM)-style [46] iterations. At a given level, each region is considered separately from the others in some arbitrary order. A spatial cutline for the region, either horizontal or vertical, can be carefully chosen. Given some initial partition, subsets of cells are moved across the cutline to reduce the total weight of hyperedges cut without violating a given balance constraint. This constraint can be set loosely initially and then gradually tightened. As the recursion proceeds, cell subsets become smaller, and the cell-area distribution over the placement region becomes more uniform. A small example of partitioning-based placement is shown in Figure 31.

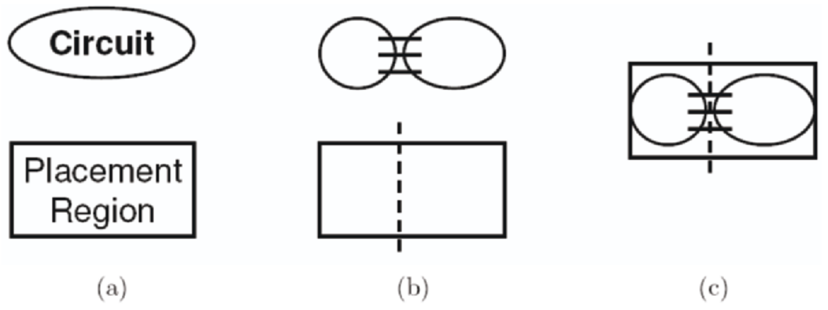


Figure 31. An example of partitioning-based placement (a) Given circuit and placement region. (b) Partition the circuit and find the corresponding cutline. (c) Assign partitions into subregions

A.2.1. Multilevel Partitioning

Traditional graph partitioning algorithms compute a partition of a graph by operating directly on the original graph. These algorithms are often too slow and/or produce partitioning solutions. Multilevel partitioning algorithms, on the other hand, have been shown to be very scalable and effective [68]. These algorithms, as illustrated in Figure 32, consist of three phases.

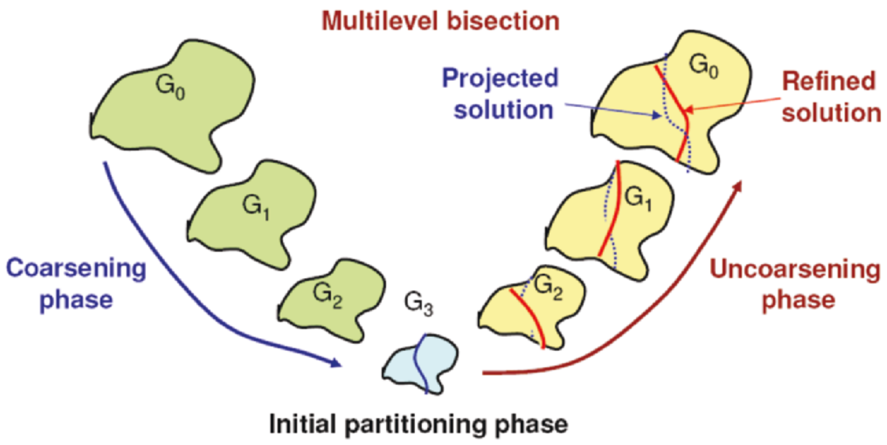


Figure 32. Multilevel partitioning framework

- Coarsening Phase. The purpose of coarsening is to create a smaller hypergraph, such that a good bisection of the smaller hypergraph is not significantly worse than the bisection directly obtained for the original hypergraph. It recursively groups together vertices based on some connectivity metric (each vertex is highly connected with at least one other vertex in the group) level by level until the number of vertices is less than a threshold. At each level, hypergraph coarsening helps reducing the size of the hyperedges. That is, after several levels of coarsening, large hyperedges are contracted to hyperedges connecting just a few vertices.
- Initial Partitioning Phase. After a small hypergraph is obtained, we may apply the FM [46] heuristic or even balanced random bisection to obtain an initial partitioning result efficiently.
- Uncoarsening Phase. During the uncoarsening phase, it declusters the hypergraph while applying a partitioning refinement algorithm (e.g., the FM heuristic) to improve the quality level by level. At each level, a partitioning of the coarser hypergraph is used to obtain a partitioning for the finer graph. This is done by successively

projecting the partitioning to the next-level finer hypergraph and using a partitioning refinement algorithm to reduce the cut and thus improve the quality of the partitioning. Since the next-level finer hypergraph has more degrees of freedom, such refinement algorithms, say the FM heuristic, tend to improve the quality.

A.2.2. Terminal Propagation

Connections between subregions can be modelled by terminal propagation [41], in which the usual cutsize objective is augmented by terms incorporating the effect of connections to external subregions. Figure 33 shows the effect of the terminal propagation. A proper terminal propagation leads to a better solution.

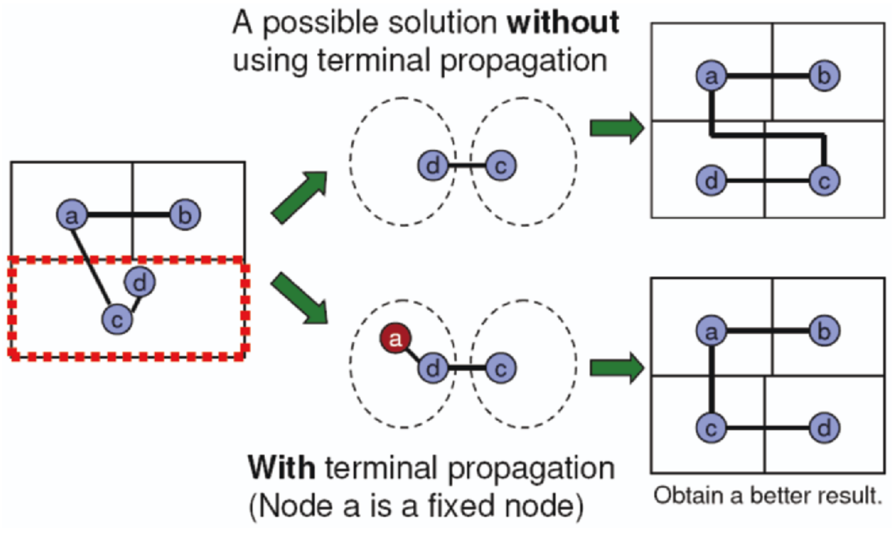

A possible solution **without** using terminal propagation
a
b
c
d
With terminal propagation (Node a is a fixed node)
Obtain a better result.

Figure 33. An example of the effect of terminal propagation

Since the net weight in the traditional terminal propagation for the min-cut based placement is a constant value, the weight with the change in HPWL cannot be exactly modelled, whether a net is cut or not. The underlying idea for exact net-weight modeling (terminal propagation) presented by Chen, Chang, and Lin in [21] is that we want to map the min-cut cost exactly to the HPWL change. Another net-weighting method was proposed in [25]; they discussed the net-weighting method for partitioning based on four cases. However, they can obtain exact modeling only for two-terminal nets, i.e., they can only obtain suboptimal results for multi-terminal nets. Unlike the previous work that exhaustively enumerates of potential cases, the following unified model presented in [21] assigns the net weights to map the HPWL value exactly. The HPWL modeling not only can be applied to vertical-cut or horizontal-cut partitioning, but can also be applied to placement feedback (repartitioning) [63]. Further, the unified HPWL model can even apply to the partitioning associated with two non-adjacent regions.

A circuit is modelled as a hypergraph. Each node in the hypergraph corresponds to a block inside the target region, with the node weight being set to the area of the corresponding block. Each hyperedge denotes a two- or multi-terminal net in the circuit, with the hyperedge weight being set to the value of the HPWL contribution if the hyperedge is cut.

Let w_1 be the HPWL when all blocks are at the side closer to the span of the terminals, w_2 be the HPWL when all blocks are at the *opposite* side, and w_{12} be the HPWL when blocks are at the both sides. See Figure 34 for an illustration. Let n_{cut} be the cutsize of the net for the corresponding hypergraph. So, we have $w_{12} \geq w_2 \geq w_1$. Then, we introduce a partitioning hypergraph with two fixed nodes to represent the two sides and movable nodes to represent the movable blocks. We then add two hyperedges e_1 and e_2 into the hypergraph. The hyperedge weight can be determined as follows. We introduce e_1 to connect the fixed node corresponding to the side closer to the span of terminals and all movable nodes and e_2 to connect between all movable nodes. We then assign the weight of the hyperedge e_1 as the value $w_2 - w_1$ (note that $w_2 \geq w_1$), and that of the hyperedge e_2 as the value $w_{12} - w_2$. Partitioning the resulting hypergraph can determine to which partition the block belongs. Based on the exact net-weight model, we have the following theorems [21]:

Theorem 1 *With the unified net-weight modeling, we have HPWL* $= w_1 + n_{cut}$.

Theorem 2 *The unified net-weight modeling exactly maps HPWL to the min-cut cost.*

B. Cell Legalization

NTUplace extends the method proposed in [64] to handle cell legalization. Cells are sorted according to their coordinates, and then

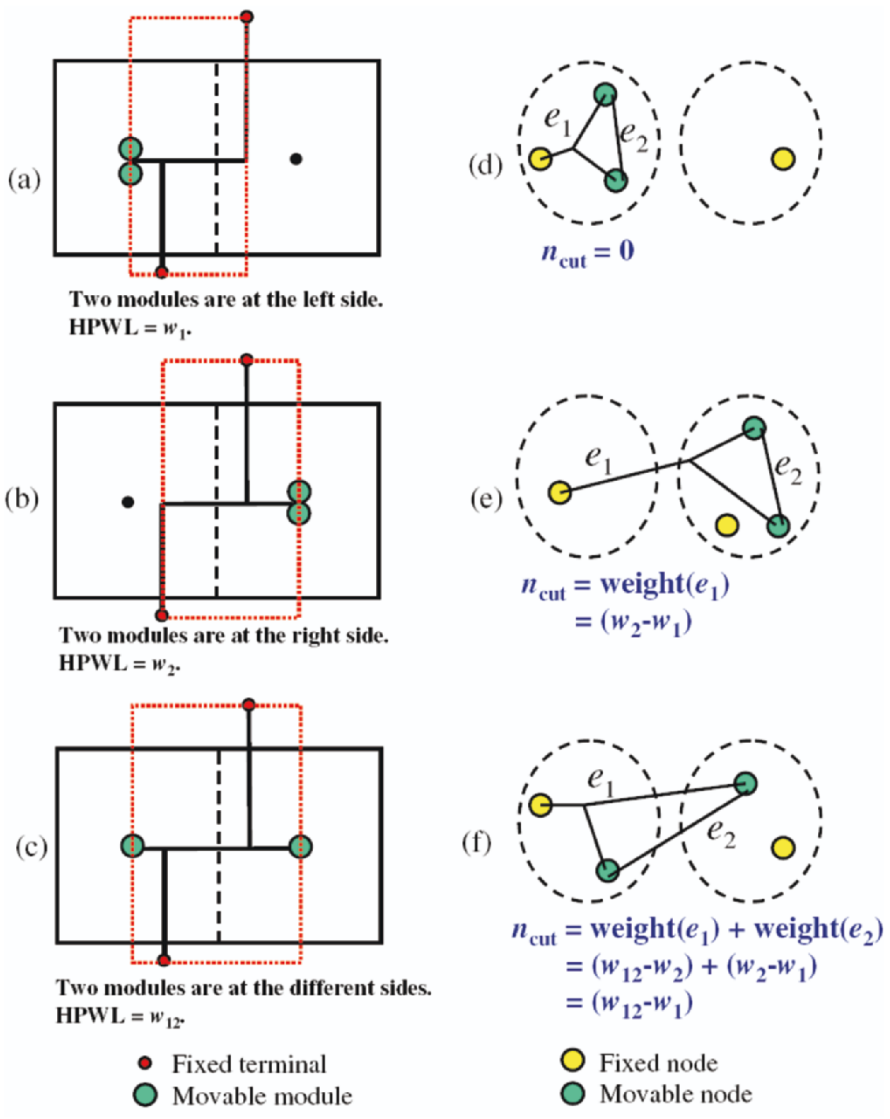

(a)
Two modules are at the left side.
HPWL = w_1.
(b)
Two modules are at the right side.
HPWL = w_2.
(c)
Two modules are at the different sides.
HPWL = w_{12}.
(d)
e_1
e_2
$n_{cut} = 0$
(e)
n_{cut} = weight(e_1) = (w_2-w_1)
(f)
n_{cut} = weight(e_1) + weight(e_2) = (w_{12}-w_2) + (w_2-w_1) = (w_{12}-w_1)
Fixed terminal
Movable module
Fixed node
Movable node

Figure 34. An example of determining a net weight. (a), (b), and (c) are three possible partitioning results. (d), (e), and (f) are corresponding partitioning hypergraphs

each cell is placed to the closest available position. In addition to the sorting in left-to-right ordering and right-to-left ordering, we add a center-to-two-sides ordering starting from the most congested column, which sorts all cells according to their distance to the chip center. Finally, we take the best among the three legalization results.

C. Detailed Placement

In the detailed placement stage, three techniques are applied to improve the final placement result: (1) window-based detailed placement, (2) branch-and-bound cell swapping, and (3) horizontal whitespace distribution.

In window-based detailed placement (*WDP* for short), it first creates windows according to the given window size and the overlap range between two windows. In each window, WDP finds a group of exchangeable cells. The cost of assigning each cell to a legal position (e.g., the displacement) is calculated, and a transportation formulation is applied to find an optimal assignment [38]. In their implementation, they iteratively change the window size and the overlap range between two windows to perform WDP until no significant improvement is achieved or the given runtime limitation is met.

For branch-and-bound cell swapping, it selects k cells each time to find the best ordering of cells by enumerating all possible orderings using the branch-and-bound algorithm. Here, k is a user-specified parameter. This process is repeated until all standard cells are processed.

Horizontal whitespace distribution optimally arranges the whitespace in a row without changing the cell ordering [62,65]. Given m ordered cells and a row of width n, an $m \times n$ table is constructed and the optimal positions of each cell can be determined through a dynamic programming algorithm. For the placer, horizontal whitespace distribution is applied row by row to optimize the cell locations.

D. Other Techniques for Partitioning-Based Placement

Careful consideration of the order and manner in which subregions are selected for partitioning can be significant. For example, a dynamic programming approach to cutline selection can improve overall results by 5% or more [110]. In the multi-way partitioning framework, intermediate results from the partitioning of each subregion are used to influence the final partitioning of others. Explicit use of multi-way partitioning at each stage can in some cases bring the configuration closer to a global optimum than is possible by recursive bisection alone [109]. Cell replication and iterative deletion have been used for this purpose. Rather than attempting to find the best subregion in which to place a cell, we can replicate the cell to place it once in every subregion, then iteratively delete only the worst choices. These iterations may continue until only one choice remains, or they may be terminated earlier, allowing a small pool of candidates to be propagated to and replicated at finer levels. By postponing further deletion decisions until better information becomes available, spurious effects from locally optimal subregion partitions can be diminished and the global result improved.

The partitioning solution can be improved by combining min-cut objective with an analytical (quadratic programming) technique. For each partitioning region, we use springs to model the connectivity of the circuit. The total potential energy of those springs is a quadratic function of their length. An initial placement solution can be obtained by solving the quadratic placement problem. According to the initial placement solution, the cells far from the centerline of the partitioning region are temporarily fixed during this level of global placement (see Figure 35). The fixed cell locations also provide other partitioning regions with more accurate terminal propagation information than traditional terminal propagation which assumes cells to be placed in the center of their regions.

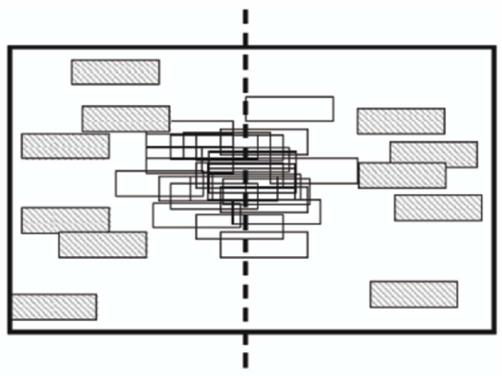


Figure 35. An example of the cell positions inside a partitioning region after wirelength optimization by quadratic programming. Cells in gray will be fixed at the corresponding regions during this level of partitioning

3.3.2 Simulated annealing based placement

Perhaps the best known simulated-annealing based placement algorithm is TimberWolf [96]. It consists of two stages. At Stage 1, blocks are moved between different rows as well as within the same row. Blocks overlap are allowed at this stage, and will be removed at the second stage. When the temperature is reached below a certain value, Stage 2 begins. At Stage 2, we remove any overlaps and continue the annealing process, but only interchange adjacent blocks within the same row. The solution perturbations are based on three types of moves:

- M1: Displace a block to a new location.
- M2: Interchange two blocks.
- M3: Change the orientation of a block.

TimberWolf first tries to select a move between M1 and M2 with the probabilities 0.8 and 0.2 for M1 and M2, respectively. If a move of type M1 is chosen and it is rejected, then a move of type M3 for the same block will be chosen with the probability 0.1. TimberWolf applies a range limiter (window size) to define the row that a block can be displaced and the pairs of blocks that can be interchanged. At the beginning, the width and height of the window is big enough to contain the whole chip. The window size shrinks proportionally to $\log(T)$ as the temperature T decreases. Stage 2 begins when the window size is so small that no inter-row block interchanges are possible. TimberWolf can handle up to tens of thousands of blocks well. With millions of blocks/cells in modern SOC design, a state-of-the-art placement algorithm that can deal with large-scale circuit sizes is desired.

In the following, we introduce a more recent simulated-annealing based placement tool, called Dragon, presented in [101]. Figure 36 shows its placement flow. Dragon integrates the partitioning and simulated-annealing techniques to cope with large-scale placement. A circuit is recursively partitioned alternatively along horizontal and vertical cut lines. The subcircuits after partitioning are assigned to rectangular bins. The bin-based simulated annealing that moves blocks in the bins is performed to improve the current placement solution. Such a procedure terminates when a certain stop criteria (e.g., average number of cells per bin is less than a given number) is met. An adjustment step is then executed to fit the bin-based placement into row structures. The next step is a cell-based simulated annealing. The bin structure still exists and the cells are moved between the centers of bins. The locations of these centers can be changed during annealing. The final step simply spreads overlapped cells and makes local improvements to obtain the detailed placement.

To handle the high complexity of the problem, the input netlist is recursively divided into two partitions using a state-of-the-art min- cut partitioner, hMetis [68], such that the number of cuts across the partitions is minimized and the sizes of two partitioned sets satisfy some predefined balance constraint.

A major drawback of the min-cut, partitioning based placement is its irreversibility. Once a cell is assigned to one side of the cut line, it will never move to the other side to improve the placement. Multilevel simulated annealing is applied to help placements move out of the local minima. The key idea is to reduce the number of movable objects in annealing. Low-temperature annealing is adopted at each level and the number of levels is not fixed. At the final placement stage, a fast greedy improvement is used to speed up the process. Both bin annealing and cell annealing uses the same cost function of total wirelength and the same cooling schedule. Swapping is the main move in both types of annealing

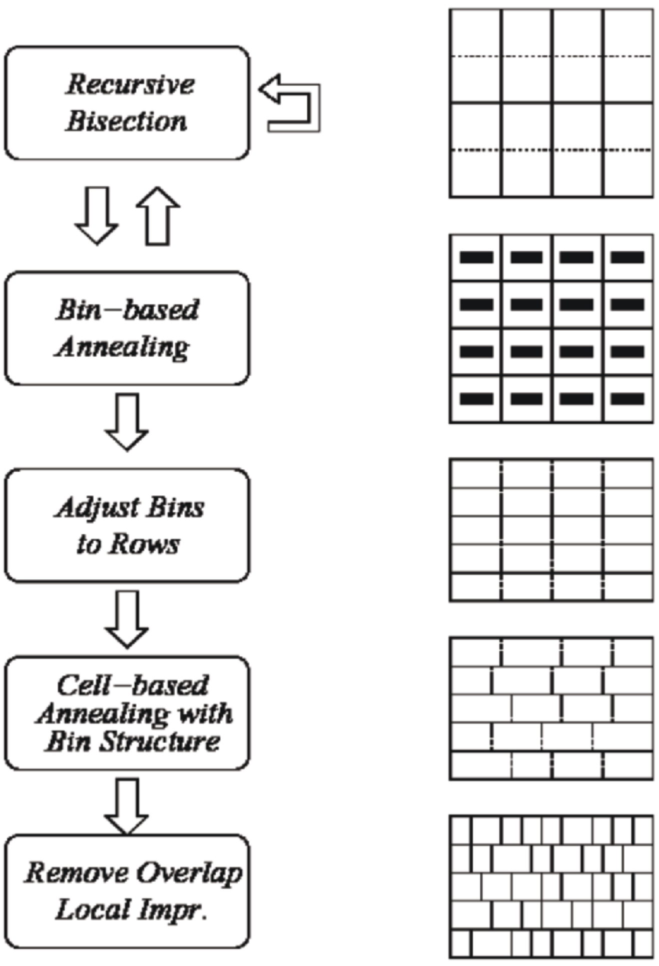

Recursive Bisection
Bin–based Annealing
Adjust Bins to Rows
Cell–based Annealing with Bin Structure
Remove Overlap Local Impr.

Figure 36. The Dragon placement flow

while shifting is used a little bit in cell annealing. Although it tries to reduce the time-consuming annealing by bin-based approach, the running-time cost is still very high.

3.3.3 Analytical placement

A force-directed method for global placement was introduced in [42]. The global placer is named Kraftwerk. In addition to the well-known wirelength dependent forces, Kraftwerk uses additional forces to reduce cell overlaps and to consider the placement area. The wirelength dependent quadratic objective function to minimize is described as follows. Let *n* be the number of movable cells in the circuit and (x_i, y_i) be the coordinates of cell i. A placement of the circuit can be described by the $2n$-dimensional vector $\vec{p} = (x_1, ..., x_i, ..., x_n, y_1, ..., y_i, ..., y_n)^T$. The circuit connectivity is modelled as a graph. Cells are modelled as vertices, nets are modelled as edges, and hyperedges are modelled as cliques. The cost of an edge is modelled as the squared Euclidean distance between its adjacent vertices multiplied with the weights of the edges. The squared Euclidean distance between cells i and j is $(x_i - x_y)^2 + (y_i - y_j)^2$. The objective function sums up the cost of all edges and can be written in matrix notation as

$$\frac{1}{2}\vec{p}^T C\vec{p} + \vec{d}^T\vec{p} + const. \tag{17}$$

This objective function is minimized by solving the linear equation system

$$C\vec{p} + \vec{d} = 0. \tag{18}$$

Additional constant forces are introduced in [42] to distribute the cells more evenly in the layout region.

$$C\vec{p} + \vec{d} + \vec{e} = 0. \tag{19}$$

The force vector $\vec{e}$ contains additional forces working on each cell in the x and y directions. These additional forces try to move the cells from high-density regions to low-density regions in the layout, thus attempting to reduce the overlaps. The algorithm described in [42] is iterative and determines the additional forces according to the current placement. In each iteration, the forces acting on the cells are assumed constant and are used to calculate a new placement. The new placement is the base for the next iteration step and so on. Each step of the algorithm is called a placement transformation. The transformation step can be applied to fully overlapping placements as well as nearly legal placements. Thus, the algorithm renders itself very elegantly to ECO style placement requirements.

It is argued in [42] that their algorithm is able to handle large mixed-size placement problems without treating macros and standard cells differently. However, if applied from scratch on constrained mixed-size designs with less whitespace, this algorithm frequently produces placements with large overlaps [4].

Recently, the force-directed placement framework has been generalized to a more rigorous mathematical formulation and adapted to a multilevel implementation in mPL5 [15]. An overview of the mPL5 formulation is given here.

Placement objectives and constraints are approximated by smooth functions. A bounding-box weighted wirelength objective is approximated by the log-sum-exp model[44,67].

$$W(x,y)=\gamma \sum_{\text{nets} e\in E}\left(\log \sum_{\text{nodes} v_k\in e} e^{x_k/\gamma}+\log \sum_{\text{nodes} v_k\in e} e^{-x_k/\gamma}+\log \sum_{\text{nodes} v_k\in e} e^{y_k/\gamma}+\log \sum_{\text{nodes} v_k\in e} e^{-y_k/\gamma}\right), \tag{20}$$

where x and y denote the vectors of cell's x - and y -coordinates. The smaller the parameter γ, the more accurate the approximation. Letting $D+ij$ denote the cell area density of bin B_{ij} and K the total cell area divided by the total placement area, the area-density constraints are initially expressed simply as $D_{ij}=K$ over all bins B_{ij}. Viewing the D_{ij} as a discretization of the smooth density function $d(x,y)$, these constraints are smoothed by approximating d by the solution ψ to the Helmholtz equation

$$\begin{cases} \Delta\phi-\varepsilon\psi(x,y)=d(x,y), & (x,y)\in R \\ \dfrac{\partial\psi}{\partial v}=0, & (x,y)\in\partial R \end{cases} \tag{21}$$

where $\varepsilon>0$, v is the outer unit normal, ∂R is the boundary of the placement region R, $d(x,y)$ is the continuous density function at a point $(x,y)\in R$, and Δ is the Laplacian operator $\Delta\equiv\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}$. The smoothing operator $\Delta_\varepsilon^{-1}d(x,y)$ defined by solving Equation (21) is well defined, because Equation (21) has a unique solution for any $\varepsilon>0$. Since the solution of 24 has two more derivatives [45] than $d(x,y)$, ψ is a smoothed version of d. Discretized versions of 21 can be solved rapidly by fast multilevel numerical methods. Recasting the density constraints as a discretization of ψ gives the nonlinear programming problem

$$\begin{array}{ll} \text{Minimize} & W(x,y) \\ \text{Subject to} & \psi_{ij}=-K/\varepsilon, 1\le i\le m, 1\le j\le n, \end{array} \tag{22}$$

where ψ_{ij} is obtained by solving Equation (24) with the discretization defined by the given bin grid. This nonlinear-programming problem is solved by the Uzawa iterative algorithm [8], which does not require second derivatives or large linear-system solves:

$$\begin{gathered} \nabla W(x^{k+1}, y^{k+1}) + \sum_{i,j} \lambda_{ij}^k \nabla \psi_{ij} = 0 \\ \lambda_{ij}^{k+1} = \lambda_{ij}^k + \alpha(\psi_{ij} + K/\varepsilon) \end{gathered} \tag{23}$$

where λ is the Lagrange multiplier, $\lambda^0 = 0$, α is a parameter to control the rate of convergence, and gradients of ψ_{ij} are approximated by simple forward finite differences $\nabla_{x_k}\psi_{ij} = \frac{\psi_{i,j+1}-\psi_{i,j}}{h_x}$, $\nabla_{y_k}\psi_{ij} = \frac{\psi_{i+1,j}-\psi_{i,j}}{h_y}$ when the center of cell v_k is inside B_{ij} and is set to zero otherwise. The nonlinear equation for (x^{k+1}, y^{k+1}) is recast as an ordinary differential equation and solved by an explicit Euler method [87].

3.3.4 Combining floorplanning and placement for mixed-size designs

The work in [3,5] proposes a three-stage floorplanning/placement flow to handle mixed-size placement. First, they shred the macros into small fake cells connected by fake wires with sufficient high weights. Then the Capo [13] standard cell placer is used to obtain an initial placement. Second, the initial location of a macro is the average of respective fake cells. The small standard cells are clustered into soft blocks to perform fixed-outline floorplanning with macros. Finally, all macros are fixed and the soft blocks are decomposed into small cells. Capo is used again to place small cells.

Recent improvements to Capo include the incorporation of fixed-outline floorplanning to improve the handling of large macro blocks in mixed-size placement [2]. Min-cut placement proceeds as described above until certain ad-hoc tests suggest that legalization of a subset of blocks and cells within their assigned subregion may be difficult. At that point, the cells in that subregion are aggregated into soft clusters, and annealing-based fixed-outline floorplanning is applied to the given subproblem [3]. If it succeeds, the macro locations in its solution are fixed. If it fails, it must be merged with its sibling subproblem, and the merged parent subproblem must then be floorplanned. This step therefore initiates a recursive backtracking through ever larger ancestor subproblems. The backtracking terminates as soon as one of these ancestor subproblems is successfully floorplanned. The ad-hoc tests are chosen to prevent long backtracking sequences on most test cases, as the floorplanner does not scale well to large subproblems. Adya et al. [2] observe that it is typically possible to define the ad-hoc tests so that the transition from min-cut partitioning to fixed-outline floorplanning does not impair scalability. However, as the algorithm cannot ensure the legalizability of the subproblems it generates by min-cut partitioning, it cannot prevent the possibility of a long backtracking sequence or a failure, especially on difficult low-whitespace instances.

3.4 Timing-Driven Placement

Timing-driven placement algorithms can be classified into two major categories: (1) path-based and (2) net-based methods. The path-based algorithms try to control critical path delays directly, and the net-based methods transfer the timing constraint of each path into the weight of each net.

3.4.1 Path-based algorithms

Typical methods in this category consist of two steps: (1) formulate the problem as a linear or non-linear programming problem by introducing auxiliary variables [50,58,99], and (2) minimize the length of a set of critical paths [100]. All path-based algorithms share the advantage of a more accurate control over the timing of the critical paths. However, they usually require substantial computation resources due to the exponential number of paths that need simultaneous optimization.

3.4.2 Net-based algorithms

In net-based algorithms, timing constraints are first translated into net weights [12,40] or delay budgets [89,94,107,112]. Net weights are used to distinguish timing-critical nets (assigned with larger weights) from non-critical ones (assigned with smaller weights). The weakness of this approach is that the net weights alone cannot control the placement results well. The goal of delay budgeting is to assign the allowable delays or constraints on individual nets such that the target timing can be met if all the constraints are satisfied. In practice, due to the intractability of placement problems and the way the constraints are assigned, it is possible that a placer might not find a feasible solution which satisfies all the constraints. It is often the case that some nets are assigned unnecessarily large budgets while others' budgets may be slightly changed. The reason is that initially the delay-budgeting process lacks a clear picture of the final placement. Therefore, it often cannot budget delays accurately on individual nets.

3.5 Conclusion

The traditional placement problem seeks to minimize wirelength under the constraint that cells/macros do not overlap with each other. We have introduced three types of the most popular techniques used in the state-of-the-art placers: (1) the partitioning based approach, (2) the simulated annealing based approach, and (3) the analytical approach. The partitioning based approach has great scalability for large-scale designs and is easier for cell density control. However, if the chip utilization rate is low (i.e., large deadspace), a partitioning-based placer might not minimize the wirelength well; in contrast, the analytical approach is more suitable for the instances with low utilization rate since it aims to compute the best cell locations first. The simulated annealing based approach can obtain high-quality solutions for manageable problem sizes, but it may be prohibitively time-consuming for current simulated annealing based placers to work on very large-scale designs.

In SOC designs, large-scale mixed macro and standard-cell placement and timing-driven placement are two major challenges. Many mixed-size placement algorithms are reported in the literature recently, and they can be classified into three categories: The first category places macros and standard cells simultaneously, such as APlace [66], Feng Shui [70], mPG-MS [16], mPL [15], and UPlace [108]. The second category combines floorplanning and placement techniques, such as Capo [2]. The third category works in two stages: first place the macros and then the standard cells, such as the algorithm presented in [4]. With the dramatic increase in the design complexity, more effective large-scale mixed-size placement algorithms are desirable.

For timing-driven placement, existing algorithms can be classified into two major categories of the path-based and the net-based methods. The path-based algorithms try to control critical path delays directly, and the net-based methods transfer the timing constraint of each path into the weight of each net. The net-based algorithm has much less complexity than the path-based one. To effectively weight each net, we must consider two important issues: how many paths share this net and the criticality of it. Existing algorithms such as the PATH algorithm presented in [73] can consider these two issues, but need to use Static Timing Analysis (STA) to evaluate the net criticality. It is time-consuming and cannot handle large-scale designs efficiently. Hence, it is desirable to develop more efficient algorithms to evaluate the criticality of each net without STA.

4. ROUTING

4.1 Introduction

The continuous increasing SOC design complexity imposes severe challenges for modern router design. As pointed out in [29], a 2.5 × 2.5 cm^2 chip in the 70-nm technology may have over 360,000 horizontal and vertical routing tracks. In addition, the 90-nm technology node has design rules in the high hundreds to low thousands, whereas the forthcoming 65-nm node may have several thousand design rules. To tackle the challenges, the routing frameworks are evolving from the *flat* framework to the *hierarchical* and *multilevel* frameworks. We detail the three routing frameworks in the following.

4.2 Flat Routing Framework

Routing is typically a very complex problem. In order to make it manageable, a traditional routing system usually uses the two-stage flat framework of *global routing* followed by *detailed routing*. Global routing first partitions the entire routing region into tiles (or channels) and decides tile-to-tile paths for all nets while attempting to optimize some specified objective functions (e.g., the total wirelength and the critical timing constraints). Then, guided by the results of global routing, detailed routing determines actual tracks and vias for all nets according to the design rules.

Many routing algorithms adopt this two-stage flat framework. These algorithms can be classified into *sequential* and *concurrent* approaches.

4.2.1 Sequential approach

Perhaps the most straightforward strategy for routing is to select a specific order and then to route nets sequentially in that order. The main advantage of this approach is that the congestion information for previously routed nets can be taken into consideration while routing a given net. The drawback of sequential approach is that the quality of the routing solution greatly depends on the order, and it is hard to find a good net ordering. In [1], Abel has concluded that there is no single net ordering technique that performs better than any other ordering method in all routing problems. Since the net ordering problem may cause unroutable nets, a rip-up/reroute procedure is often used to refine the solution.

One basic subproblem in sequential routing is to find a path connecting two pins in the presence of wiring blockages. Many algorithms have been proposed for this subproblem, and these algorithms can be classified into *maze-searching* and *line-searching* approaches.

A. Maze searching

Lee [75] proposed the first maze-searching algorithm, which adopts a two-phase approach of *wave propagation* followed by *retrace*. In the wave propagation phase, starting from the source vertex S, the accumulated length from the source to each vertex is labelled one by one according to the wavefront until the target vertex T is reached. The shortest length path is then traced back from T to S in the retrace phase. Figure 37 illustrates the process of Lee's maze-search algorithm. The Lee's algorithm guarantees to find a connection between two terminals if it does exist and the connection is the shortest path. However, in practice, the maze-searching algorithm is slow and has large memory requirements; therefore, it cannot be applied to large designs directly.

Many efforts have attempted to improve its speed and memory usage. Akers [6] proposed a *coding sequence* technique to reduce memory requirements. Instead of wavefront numbers, Hadlock [49] used detour numbers for wave labelling to substantially reduce the search space and running time. Soukup [98] combined breadth-first search and depth-first search approaches to the wave propagation; with this approach, the maze-searching algorithm can speed up 10–50 times than Lee's algorithm, but the disadvantage is that it does not guarantee to find the shortest path. Some techniques such as *starting point selection*, *double fan-out*, and

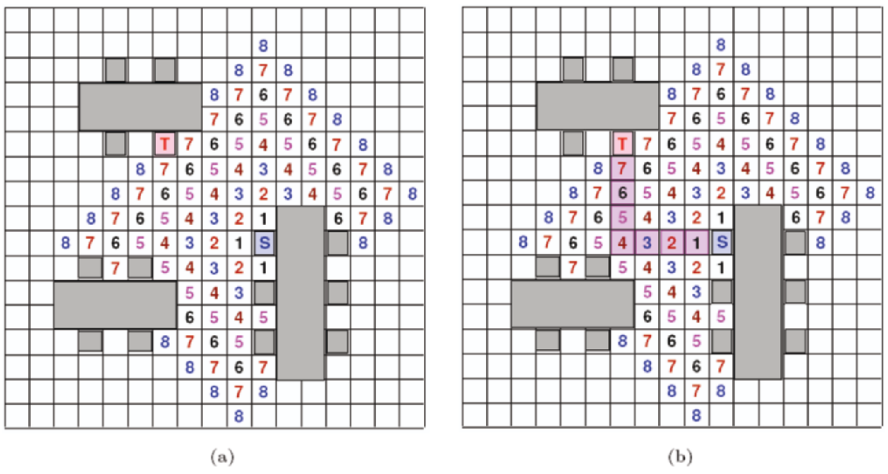


Figure 37. Lee's maze-search algorithm. (a) Wave propagation. (b) Retrace

framing are proposed to reduce the search space of wave propagation and therefore to speed up the running time considerably [92].

Although there exists some disadvantages in this maze-search method, the maze-search approach still plays an important role and is usually incorporated into other existing routing algorithms. For example, Cong and Madden [33] integrated maze routing and the *iterative deletion* technique to develop a performance-driven multilayer area router for printed circuit board (PCB) and multi-chip block (MCM) designs.

B. Line searching

As mentioned earlier, the major drawbacks of the maze-searching algorithm are the high amount of memory required and long running time. The line-search algorithm overcomes these drawbacks by using line segments to represent the routing space and paths.

Mikami and Tabuchi [86] proposed the first line-search algorithm. As opposed to the maze-searching algorithm, which mainly proceeds in a breadth-first manner, the line-searching algorithm performs a depth-first search. The line-searching algorithm initially sets the source S and the target T as *base points*, and then generates four (two horizontal and two vertical) line segments passing through these base points. These line segments are extended until they hit the design boundary or obstacles. Then, the intersections of these line segments are iteratively set as new base points, and four new line segments are generated for these new base points. This process repeats until a segment generated from S intersects a segment generated from T, and a connection can be found by tracing from

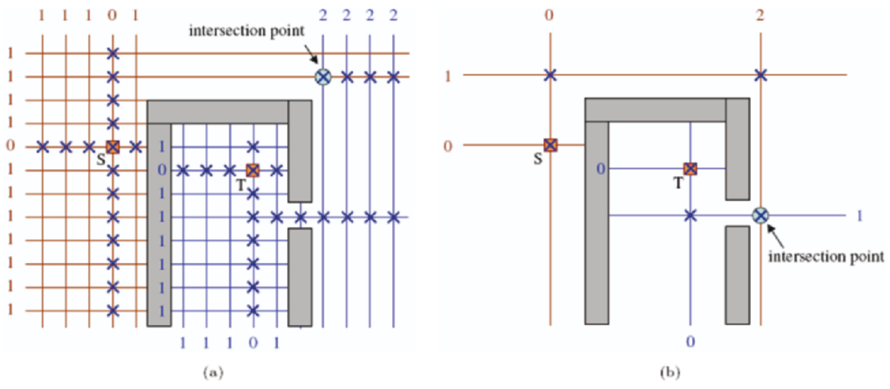


Figure 38. Line-searching algorithms. (a) Mikami-Tabuchi's algorithm. (b) Hightower's algorithm. The crossing points denote the base points, and the numbers denote the sequence of the search process

this intersection point to both S and T. Figure 38(a) gives an example of the Mikami-Tabuchi's line-searching algorithm. Like Lee's maze-searching algorithm, Mikami-Tabuchi's line-searching algorithm also guarantees to find a path if one exists, but it may not always be the shortest. The line-searching technique significantly reduces both memory requirements and execution times.

Later, Hightower [52] proposed another line-searching algorithm, which is similar to Mikami-Tabuchi's algorithm. The difference is that Hightower's algorithm only considers those line segments that are extendable beyond obstacles, and each line segment has at most two base points. Figure 38(b) illustrates Hightower's line-searching algorithm. Because fewer line segments are considered, Hightower's algorithm has more dramatic memory saving than Mikami-Tabuchi's algorithm. However, Hightower's algorithm might fail to find a path even if one exists. To remedy the deficiencies, it needs *backtracking* procedures to choose the right base points, and therefore, the running time does not improve very much more than Lee's maze-searching algorithm in practice.

4.2.2 Concurrent approach

The major drawback of the sequential approach is that it suffers from the net-ordering problem. Under any net ordering, it is more difficult to route the nets that are considered later because they are subject to more blockages. In addition, if the sequential routing fails to find a feasible solution, it is not clear whether this is because of no feasible solution existing or because of a bad selection of net order. Moreover, when the sequential routing does find a feasible solution, we do not know whether or not this solution is optimal, or how far it is from the optimal solution. These questions may be answered if we solve the routing problem with the concurrent approach.

One common concurrent approach is to formulate global routing as a *0-1 integer linear programming* (0-1 ILP) problem. The layout is first modeled as a routing graph $G(V,E)$, where each node represents a tile and each edge denotes the boundary between two adjacent tiles. Each edge $e \in E$ is assigned a capacity, denoted by c_e, which represents the number of tracks belonging to that boundary. Given a net, all of its possible routing patterns can be enumerated. Let the variable $x_{i,j} \in \{0,1\}$ indicate if the routing pattern $R_{i,j}$ is selected from the set of routing patterns R_i of net N_i. Consequently, for a routing graph $G(V,E)$ with netlist N, the congestion-driven global routing can be formulated as a 0-1 ILP problem as follows:

$$\begin{array}{lll} \text{Minimize} & \hat{\lambda} & \\ \text{Subject to} & \sum_{R_{i,j}\in R_i} x_{i,j} = 1, & \forall N_i \in N \\ & x_{i,j} = \{0,1\}, & \forall N_i \in N, \forall R_{i,j} \in R_i \\ & \sum_{i,j:e\in R_{i,j}} x_{i,j} \le \hat{\lambda} c_e, & \forall e \in E \end{array} \tag{24}$$

The first and the second constraints require that only one routing pattern can be chosen for each net, and the third constraint with the objective together ensure to minimize the maximum congestion. If a solution of $\hat{\lambda} \le 1$ exists, an optimal global routing solution (the maximum congestion is minimized) can be achieved.

Because the 0-1 ILP is NP-complete, the high time complexity greatly limits the feasible problem size. An alternative approach to this problem is to first solve the continuous linear programming (LP) relaxation, obtained by replacing the second constraint with $x_{i,j} = [0,1]$, because LP problems can be solved in polynomial time. Then, the fractional solution obtained may be transformed to integer solutions through rounding techniques such as randomized rounding [91]. However, this approximation would inevitably lose the optimality.

In practice, the 0-1 ILP concurrent routing technique is often embedded into a larger overall global routing strategy with a divide-and-conquer manner, such as solving a subproblem, where the complexity of computing the optimal solution is manageable.

4.3 Hierarchical Routing Framework

The major problem of the flat frameworks lies in their scalability for handling larger designs. To cope with the increasing complexity, researchers proposed to use hierarchical frameworks to handle the problem. The hierarchical routing frameworks use systematic divide-and-conquer approach by transforming a large and complicated routing problem into a series smaller and simpler subproblems and then proceed in a *top-down*, *bottom-up*, or *hybrid* manner.

4.3.1 Top-down hierarchical approach

Burstein and Pelavin [11] proposed the first prominent top-down hierarchical global routing framework. They recursively divide the routing regions into successively smaller sub-regions, named *super cells*, and nets at each hierarchical level are routed sequentially or concurrently and are refined in the subsequent levels. An example of global routing by the top-down hierarchical approach is illustrated in Figure 39. Figure 39(a) gives a global-routing instance with a 3-pin net. Figure 39(b) depicts the process of top-down hierarchical global routing, in which the routing region is recursively bisected into smaller super cells, and at each level, the net is routed in terms of these super cells at that level. This process is performed in a top-down manner until the super cells reduce to the actual global routing cells.

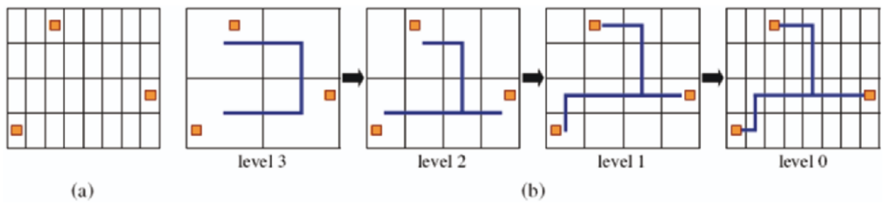


Figure 39. An example global routing using the hierarchical top-down approach. (a) A global routing instance with a 3-pin net. (b) The level-by-level top-down hierarchical global routing

Marek-Sadowska [84] proposed another top-down hierarchical framework based on a bisection and the *linear assignment* technique. When a super cell is bisected by a cut line c, any net n that must cross c is then partitioned into two subnets n_1 and n_2 by inserting a *pseudo pin* on c, such that if n crosses c through this pseudo pin, no capacity overflow would occur and the wirelength is minimized. Then, the subnets n_1 and n_2 can be solved independently in the subsequent levels. This bisection and insertion process is performed recursively in the subregions until the smallest subregions is manageable for global routing. In [84], the problem of finding a pseudo pin for each crossing net is formulated and solved as a linear assignment problem. For the global-routing instance in Figure 39(a), Figure 40 shows the bisection and pseudo-pin insertion

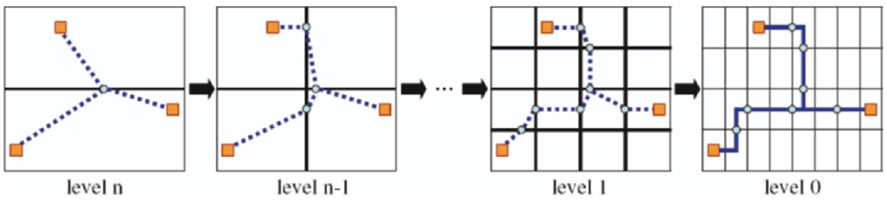


Figure 40. An example of top-down hierarchical global routing by the bisection and pseudo-pin insertion process. The dots represent the inserted pseudo pins

process at each hierarchical level. Recently, Chang *et al.* [19] also applied the linear assignment to develop a hierarchical, concurrent global and detailed router for field programmable gate arrays (FPGAs).

An advantage of hierarchical top-down approach is that the higher-level decisions are used to guide the solution at lower levels, thus reducing the net-ordering problem in sequential routing.

4.3.2 Bottom-up hierarchical approach

The first bottom-up hierarchical global routing method is described by Marek-Sadowska [83]. Initially, the routing region is partitioned into an array of 2×2 super cells. At each hierarchical level, the global routing is restrained within each super cell individually. When the routing at the current level is finished, every four super cells are merged to form a new larger super cell at the next higher level. This process continues until the top level containing only one 2×2 array is reached. Figure 41 illustrates this bottom-up approach. Figure 41(a) gives a global-routing instance with a 7-pin net. Figure 41(b) depicts the process of bottom-up hierarchical global routing, in which the solid rectangles represent the super cells, and the dots denote the merging points where two routing subsolutions of the previous level are merged together. In [57], Hu and Shing formulated the problem of finding merging points as a linear programming problem.

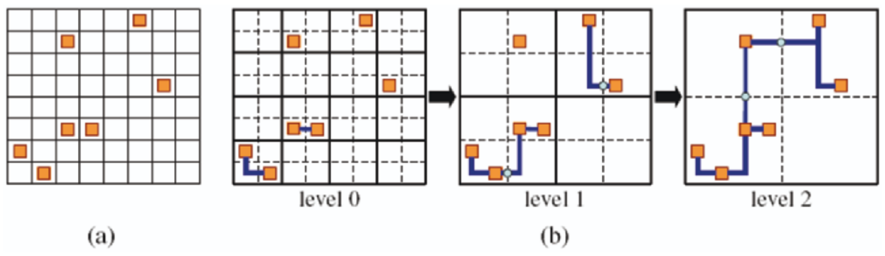


Figure 41. An example global routing using the bottom-up hierarchical approach. (a) A global routing instance with a 7-pin net. (b) The level-by-level bottom-up hierarchical global routing. The solid rectangles represent super cells, and the dots denote merging points

4.3.3 Hybrid hierarchical approach

The deficiency in the top-down and bottom-up hierarchical approaches is that the routing decision made at one hierarchical level may be suboptimal for the subsequent levels. In order to alleviate this problem, Lin *et al.* [80] proposed the first hybrid hierarchical approach that combines the bounded maze-searching algorithm with both top-down and bottom-up hierarchical methods into a unified routing framework.

Their algorithm consists of three phases: (1) neighboring propagation, (2) preference partition, and (3) bounded routing. Phase 1 performs bounded maze-searching by propagating W circles of waves out of each terminal, where W is a user-defined parameter. If the connection is not found, phase 2 recursively maps the terminal and blockages onto the adjacent upper level (Figure 42 (a)) and calls the bounded maze-search algorithm until a path is found. Then, the connected path is mapped back to the lower level to form the preferred region (Figure 42 (b)). Phase 3 performs the routing by taking the preference information into consideration (Figure 42 (c)). By means of a parameter-controlled technique, their hybrid routing demonstrates a fast speed comparable to a hierarchical router and produces routing solutions with quality similar to a maze router.

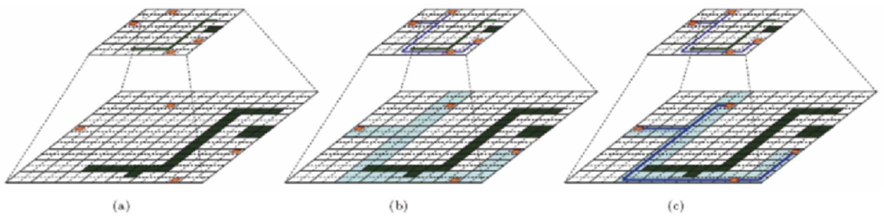


Figure 42. An example global routing using the hybrid hierarchical approach. (a) Mapping pins and blockages up one level. (b) Making connection on the upper-level and mapping down the preferred region. (c) Performing the routing within the preferred region

Later on, Hayashi and Tsukiyama [51] proposed another hybrid hierarchical global routing algorithm. The flow of their algorithm consists of two loops for the hierarchical levels, with a top-down hierarchical inner loop embedded in a bottom-up hierarchical outer loop. Specifically, the global routing mainly proceeds in a bottom-up manner, but an additional top-down refinement procedure is applied when an initial routing at each hierarchial level is obtained.

Compared with pure top-down or bottom-up hierarchical routing, the hybrid hierarchial approach has more information to generate better routing solutions.

4.4 Multilevel Routing Framework

Although the hierarchical approach can scale to larger designs, it has the drawbacks that the interactions among different routing subregions are lacking and the routing decision at a level is irreversible (i.e., cannot be refined at later stages), thus limiting the solution quality. To remedy the deficiencies, researchers have proposed various multilevel frameworks to handle large-scale routing problems. In this section, we introduce two state-of-the-art multilevel routing frameworks: (1) the Λ-shaped multilevel framework, and (2) the V-shaped multilevel framework.

4.4.1 Multilevel routing model

Both multilevel routing frameworks need to model the routing resource as a *multilevel routing graph*. At beginning, the routing region is partitioned into an array of rectangular subregions, each of which may accommodate tens of routing tracks in each dimension (see Figure 43). These subregions are usually called *global cells* (GCs). A node in the routing graph represents a GC in the chip, whereas an edge denotes the boundary between two adjacent GCs. Each edge is assigned a capacity according to the physical area or the size of a GC. This routing graph is called multilevel routing graph of level 0, denoted by G_0, where subscript represents the level.

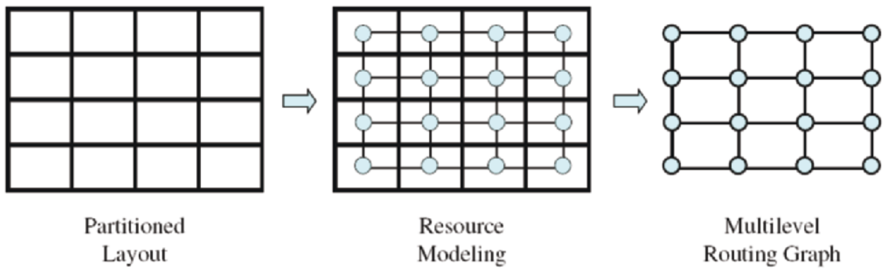
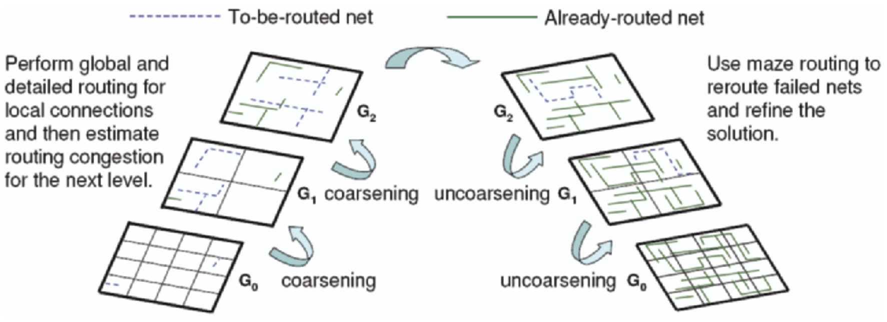


Figure 43. The multilevel routing graph

The multilevel routing algorithm consists of two stages: bottom-up *coarsening*, and top-down *uncoarsening*. The coarsening stage is a bottom-up approach that iteratively groups a set of GCs in the multilevel routing graph. This process starts from the finest level (level 0) to the coarsest level; at each level k, four adjacent GC_k of G_k are merged into a larger GC_{k+1} of G_{k+1} and at the same time perform resource estimation for use at the $k+1$ level. Coarsening continues until the number of GCs at a level is below a threshold. In contrast, the coarsening stage iteratively ungroups a set of previously clustered GCs in a top-down manner. It proceeds from the coarsest level to the finest level; at each level k, a GC_k are decomposed into four smaller GC_{k+1}.

Uncoarsening continues until the finest level is reached. Figure 44 depicts an example multilevel framework consisting of a coarsening stage followed by an uncoarsening stage.



To-be-routed net
Already-routed net
Perform global and detailed routing for local connections and then estimate routing congestion for the next level.
Use maze routing to reroute failed nets and refine the solution.
G_2
G_2
G_1 coarsening
uncoarsening G_1
G_0 coarsening
uncoarsening G_0

Figure 44. The Λ-shaped multilevel routing framework

4.4.2 Λ-Shaped multilevel routing framework

The Λ-shaped multilevel routing framework consists of bottom-up *coarsening* followed by top-down *uncoarsening*. Cong *et al.* [30,35] proposed the first Λ-shaped multilevel approach for full-chip routability-driven global routing. Their framework starts by recursively coarsening global cells, and an estimation of routing resources is computed at each level. When the coarsening is finished, a multicommodity flow algorithm is used to obtain an initial global routing solution. Then, the uncoarsening stage performs a modified maze-searching algorithm to further improve the routing solution level by level. Their experiments show better routing quality and running times than the traditional flat and hierarchical approaches.

Later, Lin and Chang [18,79] proposed an enhanced full-chip Λ-shaped multilevel global and *detailed* routing system considering both routability and performance, and their routing system show the best routability among previous works. Figure 44 illustrates their framework.

Given a netlist, they first run the minimum spanning tree (MST) algorithm to construct the topology for each net, and then decompose each net into two-pin connections, with each connection corresponding to an edge of the minimum spanning tree. At each level k during the coarsening stage, they first perform global routing for the local two-pin connections (those connections that entirely sit inside a GC_k), and then the detailed router is used to determine the exact wiring. The global routing is based on the approach used for pattern routing [69]. Let the multilevel routing graph of level 0 be $G_0 = (V_0, E_0)$ and the global routing result for a local connection c be $R_e = \{e \in E_0 \mid e \text{ is the edge chosen for routing}\}$. For the congestion control, the cost function $\alpha : E_0 \rightarrow \Re$ is applied to guide the routing:

$$\alpha(R_e) = \sum_{e \in R_e} c_e, \tag{25}$$

where c_e is the congestion of edge e and is defined by

$$c_e = 1/2^{(p_e - d_e)},$$

where p_e and d_e are the capacity and density associated with e, respectively. By dynamic density, pattern routing uses an L-shaped (1-bend) or Z-shaped (2-bend) route to make the connection, which gives the shortest-path length between two points. Therefore, the wirelength is minimum, and thus the wirelength is not included in the cost function at this stage. This cost function can guide the global router to select a path with smaller maximum congestion.

After the global routing is completed, they apply the simultaneous pathlength and via minimization (SPVM) algorithm to perform detailed maze routing to find a shortest path with the minimum number of bends/vias, if such a path exists. When the global and detailed routing are performed at level k, four adjacent GC_k are merged into a larger GC_{k+1} and at the same time resource estimation is performed for use at the next level $k+1$. Since the global routing, detailed routing, and resource estimation are integrated together at each level, the routing resource estimation is more accurate than [30,35], thus facilitating the solution refinement (e.g., the rip-up and reroute processes) at the uncoarsening stage.

Many works have been proposed to deal with different routing objectives based on this multilevel framework. Ho *et al.* [54] developed a Λ-shaped multilevel full-chip routing system with antenna avoidance, Chen *et al.* [23] presented Λ-shaped multilevel full-chip gridless routing to consider optical proximity correction (OPC) optimization, and Li *et al.* [77] applied the Λ-shaped multilevel framework to full-chip routing for testability and yield enhancement.

In [55,56], Ho *et al.* integrated an *intermediate stage* into the Λ-shaped multilevel routing framework to develop a full-chip multilevel routing system considering crosstalk optimization. The framework adopts a three-stage technique of a bottom-up congestion-driven global pattern routing stage, followed by an intermediate stage of layer/track assignment for crosstalk optimization, and then followed by a top-down point-to-path detailed routing stage. Figure 45 illustrates this framework. By performing layer/track assignment at the intermediate stage, their routing system can preserve more flexibility to allocate nets for crosstalk optimization. Later on, Ho *et al.* [53] extended this multilevel framework to the routing problems on the X-architecture.

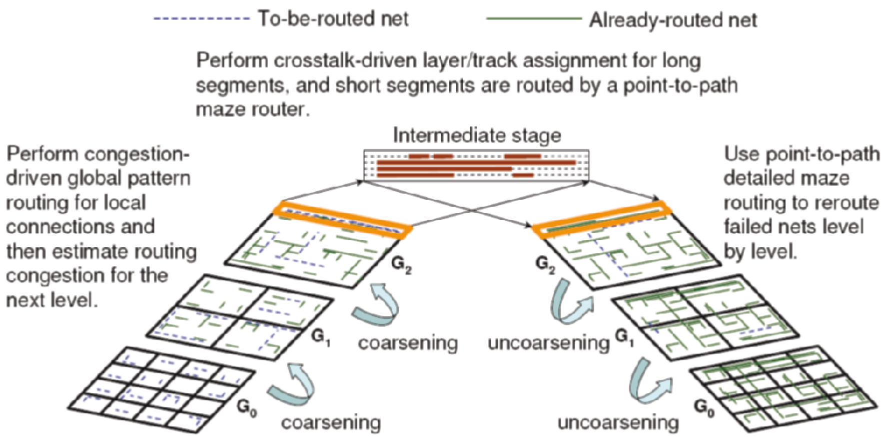

To-be-routed net
Already-routed net
Perform crosstalk-driven layer/track assignment for long segments, and short segments are routed by a point-to-path maze router.
Intermediate stage
Perform congestion-driven global pattern routing for local connections and then estimate routing congestion for the next level.
Use point-to-path detailed maze routing to reroute failed nets level by level.
G2
G1
G0
coarsening
uncoarsening
coarsening
uncoarsening

Figure 45. The Λ-shaped multilevel routing framework with an intermediate stage

4.4.3 V-shaped multilevel routing framework

Recently, Chen *et al.* [24] proposed a new V-shaped multilevel framework for large-scale full-chip gridless routing. Unlike the traditional Λ-shaped routing framework, the V-shaped one consists of top-down uncoarsening followed by bottom-up coarsening. The framework starts from the coarsest regions and then processes down to the finest ones level by level; at each level, it performs global and detailed routing and then estimates the routing resource for the next level. Then, the bottom-up coarsening stage performs global and detailed maze routing to reroute failed connections and refine the solution level by level from the finest level to the coarsest one. Figure 46 illustrates the V-shaped multilevel routing framework.

Different from the previous frameworks, they employ a dynamic *congestion map* to guide the global routing at all stages to alleviate the net-ordering problem in sequential routing. At beginning, they initialize the routing congestion information based on the pin distribution and the global-path prediction of all nets, and then keep a congestion map that is updated dynamically based on both the already-routed nets and the estimated

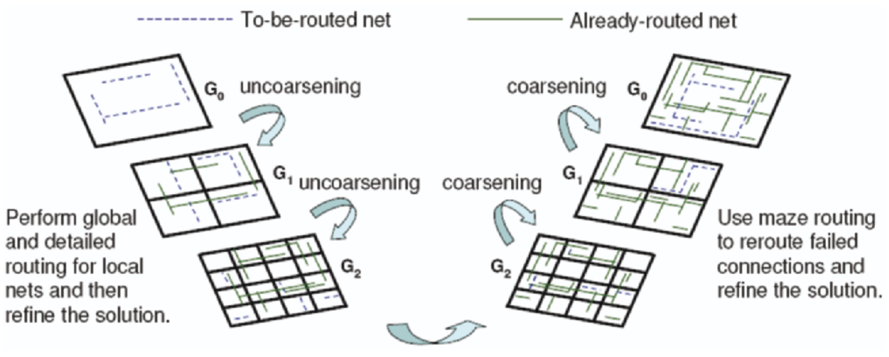

To-be-routed net
Already-routed net
G0
uncoarsening
coarsening
G0
G1
uncoarsening
coarsening
G1
Perform global and detailed routing for local nets and then refine the solution.
G2
G2
Use maze routing to reroute failed connections and refine the solution.

Figure 46. The V-shaped multilevel routing framework

unrouted nets. As routing proceeds, the congestion map is updated, and the congestion information becomes more and more accurate. Therefore, the better congestion control can be achieved throughout the whole routing process.

For a two-pin connection c, they use L- and Z-shaped pattern routes to determine the number of possible global routes n_c, and evenly distribute the wire density of the connection c, w_c, among all possible global routes. Therefore, the wire density of each possible global route equals w_c/n_c. For each possible global route, the wire density of the possible global route is added to the edge density in the multilevel routing graph. After all two-pin connections finish the process, an initial congestion map is obtained. Figure 47 gives an example of global-path congestion prediction in the congestion map. As shown in Figure 47 (a), the connection c has five possible L- and Z-shaped pattern routes from source s to target t. The number of routes passing through each global cell boundary is given in Figure 47 (b), and the congestion estimation of c in the multilevel routing graph is shown in Figure 47 (c). The experiments show that their router can

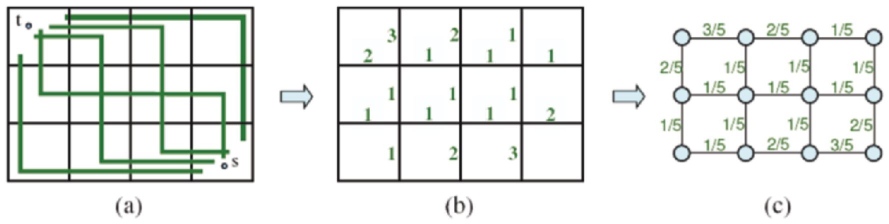


Figure 47. Global-path congestion prediction. (a) Two L-shaped and three Z-shaped pattern routes from s to t. (b) The number of routes through each boundary. (c) The pre-estimation congestion in the multilevel routing graph

obtain significantly smaller wirelength and critical path delay than the previous works.

4.4.4 Summary of multilevel routing frameworks

It has been observed that the Λ-shaped multilevel framework can handle local circuit effects (such as routability, congestion and via minimization) better since it works in a bottom-up manner and deals with local routing regions first (i.e., route shorter local nets and then longer global nets) [69,18]. In contrast, the V-shaped multilevel framework is more suitable for handling global electrical effects (such as crosstalk and critical-path delay) since it works in a top-down manner and copes with global routing regions first [24]. By performing layer/track assignment after the global routing stage, the Λ-shaped multilevel framework with an intermediate stage has more flexibilities to optimize the nanometer electrical effects; however, it is harder to accurately estimate the routing resource since the global routing and detailed routing are performed separately. Table 3 compares the properties of these multilevel routing frameworks.

Table 3. Comparison for multilevel routing frameworks

	Advantage	Disadvantage
Λ-shaped multilevel framework	More suitable to handle local effects	Harder to handle global effects
Λ-shaped multilevel framework with an intermediate stage	Flexible for addressing nanometer electrical effects	Harder to estimate routing resource
V-shaped multilevel framework	More suitable to handle global effects	Harder to handle local effects

5. METHODOLOGY SHIFT FOR SOC DESIGN

In addition to design algorithms and frameworks, design methodology is crucial for tackling the design complexity and convergence problems which are more stringent for modern SOC designs than ever. In the following, we introduce two example design methodology shifts on timing closure and power integrity arising from modern SOC designs for faster design convergence. Specifically, we introduce the design methodology problems of buffer planning for interconnect-driven floorplanning and floorplan and power/ground network co-synthesis.

5.1 Buffer Planning for Interconnect-Driven Floorplanning

5.1.1 Introduction

For deep submicron and nanometer VLSI designs, interconnection dominates overall circuit performance. However, the conventional design flow often deals with interconnection optimization at the routing or the post-routing stage. When the interconnection complexity grows drastically, it is often too late to perform aggressive interconnection optimization during or after routing since most silicon and routing resources are occupied. Therefore, it is desirable to optimize interconnection as early as possible.

Many techniques have been proposed for interconnection optimization. Some examples are wiring topology construction, buffer/repeater insertion and sizing, wire sizing and spacing [31]. Here, a buffer is composed of two inverters while a repeater is referred to as a buffer or an inverter. To simplify the discussions, we shall use buffer and repeater interchangeably throughout this chapter. Among these interconnection optimization techniques, buffer insertion is generally considered the most effective and popular technique to reduce interconnection delay, especially for global signals [7]. As an example, over 85% global nets in Intel Itanium microprocessor are buffered to reshape signals [85]. Inserting buffers in a long interconnect can break the long interconnection into shorter ones such that the overall delay can be reduced. It has been shown that without buffer insertion, the interconnection delay for a wire increases quadratically in terms of the wire length, but it increases only linearly under proper buffer insertion [9,95]. For example, it is shown in [31] that the delay of a 2cm global interconnection can be reduced in a factor of $7\times$ by optimal buffer insertion. As the intrinsic delay of a buffer becomes smaller and the chip dimension gets larger, it is expected that a large number of buffers will be inserted for modern high-performance VLSI designs (e.g., about 800K for 50nm technology [32]). With so many buffers being added, the buffer positions should be planned as early as possible to ensure timing closure and design convergence. In particular, current VLSI designs do not allow buffers to be inserted inside a circuit block since they consume silicon resource and require connections to the power/ground network. Consequently, buffers are placed in channels and dead spaces of current floorplan and are often clustered to form buffer blocks between existing circuit blocks of the floorplan, which inevitably increases the chip area [32]. It is thus desirable to carefully plan for the buffers during/after floorplanning to minimize the area overhead and facilitate routing, which is referred to as the *buffer block planning*.

However, the existence of buffer blocks imposes more design constraints. Since buffers connect global nets, the routing regions where buffer blocks are located might be congested. Further, buffers might be placed in poor locations since buffers are clustered into blocks and thus the better location for a buffer is forbidden. To remedy this deficiency, distributing buffers more uniformly in a chip naturally spreads out global nets, and thus looks promising in coping with the aforementioned problems with wire congestion and buffer blockages. In contrast to the buffer block planning methodology, as a result, Alpert et al. propose the *buffer site methodology* that allocates a buffering resource within a block by inserting a *buffer site* which can accommodate buffers (or other logic gates if not used for buffering). For *buffer site planning*, we shall plan for the buffers during/after floorplanning such that the given buffer sites can accommodate buffers and the routing timing and congestion constraints are satisfied.

To determine the optimal location for buffer insertion, we shall first consider the *feasible region* (*FR*) for a buffer, which is referred to as the maximum region where the buffer can be placed to satisfy the timing constraint. Figures 48(a) and (b) show respective FR's for inserting one and multiple buffers into a net between a source and a sink, where the FR's are shaded.

The concepts of the feasible region come in two forms. Cong, Kong, and Pan in [32] first define the "feasible region" for buffer insertion to

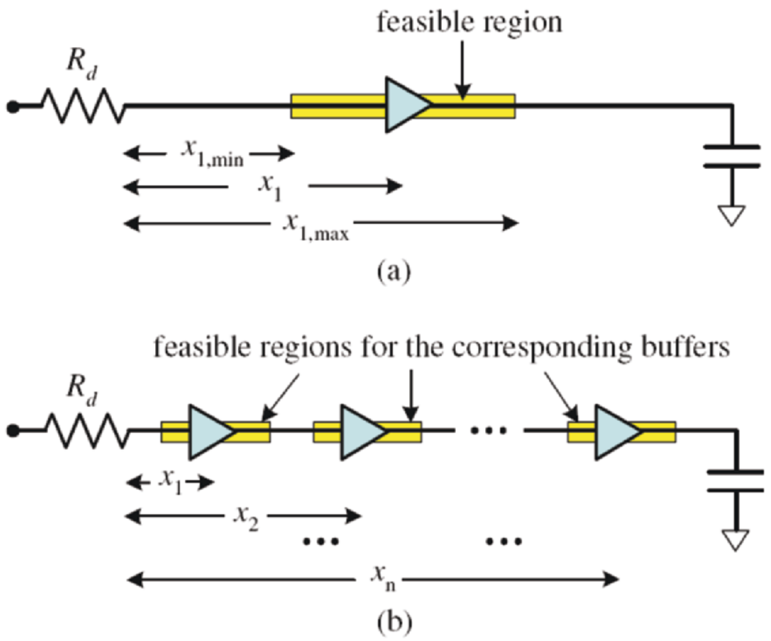


Figure 48. Feasible regions for buffer insertion. (a) Single-buffer insertion. (b) Multiple-buffer insertion

be the region where a buffer can be placed in order to satisfy a target timing constraint, assuming that all the remaining buffers are optimally placed. In contrast, Sarkar, Sundararaman, and Koh [93] introduce the idea of *independent feasible region* (*IFR*) for buffer insertion, which is defined as the region where it can be placed such that the timing constraint of the net is satisfied, assuming that the other buffers are also located within their respective independent feasible regions.

Before presenting the analytical formulae for computing the feasible regions, we shall first introduce the notation and delay model that will be used throughout this chapter. Each driver/buffer is modeled as a switch-level RC circuit [31], and each wire is modeled as a π-model. See Figure 49 for the buffer and wire models. We then use the Elmore delay model [43] for delay computation. The notation for the physical parameters of the interconnect and buffer is listed in Table 4.

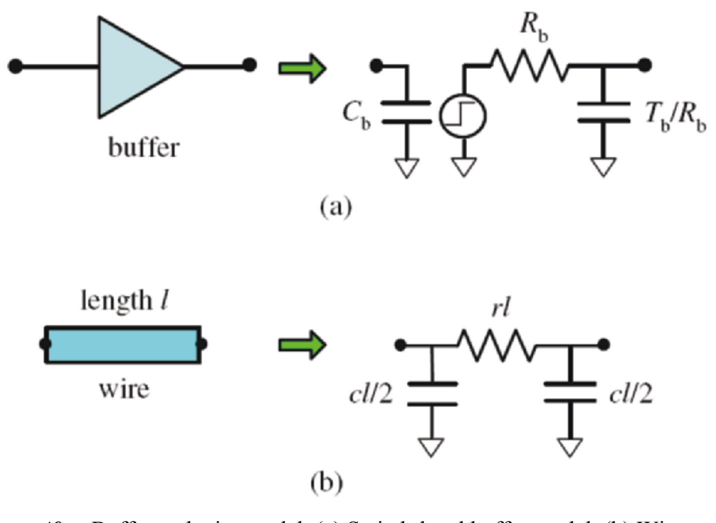


Figure 49. Buffer and wire model. (a) Switch-level buffer model. (b) Wire model

Table 4. Parameters of the interconnection and buffer

Parameter	Description
r	wire resistance per unit length
c	wire capacitance per unit length
T_b	intrinsic buffer delay
C_b	buffer input capacitance
R_b	buffer output resistance

Given a wire segment of length l with driver output resistance R and sink capacitance C, the Elmore delay of this segment is given by

$$D(R,C,l) = \left(\frac{rc}{2}\right) l^2 + (Rc + rC)l + RC. \tag{26}$$

Using the above expression, the Elmore delay of a single-source, single-sink net (i.e., two pin net) $\mathbf{N}$ of length L with n buffers can be computed by

$$D^{\mathbf{N}}(x_1, x_2, \ldots. x_n, L) =$$
$$D(R_d, C_b, x_1) + D(R_b, C_s, L - x_n) + \sum_{i=1}^{n-1} D(R_b, C_b, x_{i+1} - x_i) + nT_b,$$

where R_d is the driver resistance, C_s is the sink capacitance, and x_i is the location of the i-th buffer. The optimal locations of the n buffers for delay minimization of the net as shown in [7] are given by

$$x_i^{å} = (i-1)y_L^{å} + x_L^{å},\ i \in \{1, 2, \ldots. n\}, \tag{27}$$

where

$$x_L^{å} = \frac{1}{n+1}\left(L + \frac{n(R_b - R_d)}{r} + \frac{(C_s - C_b)}{c}\right), \tag{28}$$

$$y_L^{å} = \frac{1}{n+1}\left(L - \frac{(R_b - R_d)}{r} + \frac{(C_s - C_b)}{c}\right). \tag{29}$$

We denote the optimal delay for the net $\mathbf{N}$, of length L, with n buffers by

$$D_{opt}^{\mathbf{N}}(n, L) = D^{\mathbf{N}}(x_1^{å}, x_2^{å}, \ldots\ldots, x_n^{å}, L).$$

In the following subsection, we first discuss the computation of the feasible region and the independent feasible region on a one-dimensional line segment, and then extend the idea to the two-dimensional chip plane.

5.1.2 Feasible regions

For n buffers inserted in a two-pin net $\mathbf{N}$ as shown in Figure 48(b), [32] shows that its feasible region can be computed by the following theorem:

Theorem 3 *For a two-pin net* $\mathbf{N}$ *of the length* L *and with* n *buffers inserted and a given timing constraint* $D_{tgt}^{\mathbf{N}}$, *the feasible region for the* i *-th buffer* $(i \le n)$ *is* $x_i \in [x_{i,min}, x_{i,max}]$ *with*

$$x_{i,min} = \max\left\{0, \tfrac{K_2 - \sqrt{K_2^2 - 4K_1K_3}}{2K_1}\right\},$$
$$x_{i,max} = \min\left\{L, \tfrac{K_2 - \sqrt{K_2^2 + 4K_1K_3}}{2K_1}\right\},$$

where

$$\begin{aligned}
K_1 &= \tfrac{(n+1)rc}{2i(n-i+1)},\\
K_2 &= \tfrac{(R_b - R_d)c}{i} + \tfrac{(C_s - C_b)r + rcL}{n-i+1},\\
K_3 &= nT_b - D_{tgt}^{\mathbf{N}} + \left(R_d + (i-1)R_b + \tfrac{(n-i)rL}{n-i+1}\right)C_b + R_b((n-1)C_b\\
&\quad + C_s + cL) + \tfrac{rcL^2}{2(n-i+1)} + rLC_s - \tfrac{(i-1)c(R_b - R_d)^2}{2ir} - \tfrac{(n-i)r(C_b - C_s)^2}{2(n-i+1)c}.
\end{aligned}$$

We denote the width of the feasible region for a given buffer by W_{FR}. An analytical expression for W_{FR} is given in [32]. The following theorem presents an alternative but equivalent analytical expression.

Theorem 4 *For* $D_{tgt}^{\mathbf{N}} \ge D_{opt}^{\mathbf{N}}(n, L)$, *the width of the feasible region for the* i *-th buffer (*$i \le n$*) of the net* $\mathbf{N}$ *is*

$$W_{FR} = 2 \cdot \sqrt{\frac{2(D_{tgt}^{\mathbf{N}} - D_{opt}^{\mathbf{N}}(n, L))(n-i+1)(i)}{rc(n+1)}}.$$

5.1.3 Independent feasible regions

As opposed to the definition of feasible region, the *independent feasible region* of a buffer is the region where it can be placed while meeting the timing specifications of the net, assuming that the other buffers are placed within their respective independent feasible regions.

Formally, we define the independent feasible region (IFR) for the i-th buffer of a net $\mathbf{N}$ as

$$IFR_i = (x_i^{å} - W_{IFR}/2, x_i^{å} + W_{IFR}/2) \cap (0, L),$$

such that $\forall \quad (x_1, x_2, ..., x_i, ...x_n) \quad \in IFR_1 \times IFR_2 \times ... \times IFR_n$, $D^{\mathbf{N}}(x_1, x_2, ..., x_n, L) \le D^{\mathbf{N}}_{tgt}$. Here, W_{IFR} and $D^{\mathbf{N}}_{tgt}$ respectively denote the width of the independent feasible region IFR_i and the target delay associated with the net.

Note that the final placement of a buffer in its IFR does not depend on the placement of the other buffers, so long as they are placed within their respective IFRs. To allocate an equal degree of freedom to each buffer in the net, we choose the IFR intervals to be of equal width, which is given by the following theorem.

Theorem 5 *For* $D^{\mathbf{N}}_{tgt} \ge D^{\mathbf{N}}_{opt}(n, L)$, *the width of the independent feasible region for the* i*-th buffer (*$i \le n$*) of the net* $\mathbf{N}$ *is*

$$W_{IFR} = 2 \cdot \sqrt{\frac{D^{\mathbf{N}}_{tgt} - D^{\mathbf{N}}_{opt}(n, L)}{rc(2n-1)}}.$$

5.1.4 Two-dimensional feasible region

In the preceding discussions, we limit buffer insertion to occur along a one-dimensional line. Implicit in the discussions was the assumption that the routing from source to sink is specified by some global router. For buffer planning during floorplanning, however, no routing information is available. We typically assume that each net would be routed with a shortest path within the bounding box containing the two terminals. Therefore, we have to compute two-dimensional regions in which the buffers can be placed. The *two-dimensional feasible region* (or independent feasible region) of a buffer is defined as the union of the one-dimensional FRs (or IFRs) of that buffer on all monotonic Manhattan routes between source and sink. Therefore, 2-D FRs and 2-D IFRs are convex octilinear polygons with horizontal, vertical, and ± 1-slope boundaries (see Figure 50).

The feasible region of a buffer may be reduced by circuit blocks. Moreover, 2-D IFRs of buffers belonging to the same net are not completely independent of each other. As the widths and locations of a 2-D IFR are valid only under the assumption that a monotonic Manhattan route exists between the source and the sink, the assignments of buffers to locations within their respective 2-D IFRs should be made such that they constitute a monotone path from source to sink. In Figure 50, for example, the buffer assignments, which form a non-monotonic sequence from the source to the sink, violate the monotonicity constraint even though the buffers are within

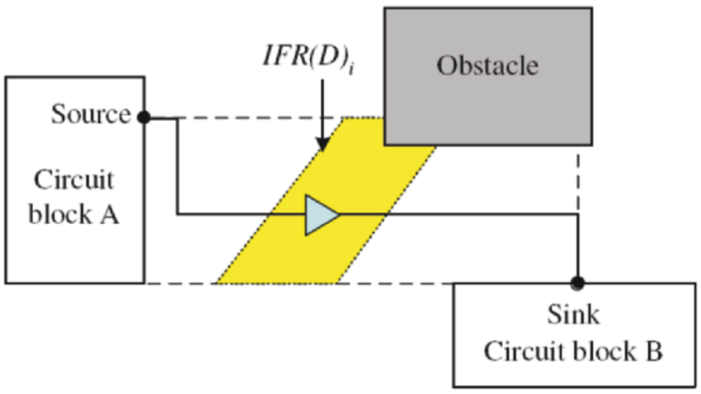


Figure 50. 2-D feasible regions and their implications on buffer assignment

their respective 2-D IFRs. Therefore, whenever the 2-D IFR of a buffer is modified, the 2-D IFRs for all other buffers in the net have to be updated if necessary.

5.1.5 Buffer block planning

The basic buffer block planning problem can be stated as follows:

- **Input:** a given floorplan (or a set of circuit blocks) and a set of nets with feasible regions for buffer insertion to satisfy the given constraints (e.g., timing)
- **Output:** the number of buffers blocks, the size and location of each buffer block, and the nets that use some buffer in this buffer block to optimize the timing.
- **Objective:** determine the size of each buffer block and its optimal location such that the overall chip area and the number of buffer blocks after buffer insertion are minimized and the percentage of the satisfied timing constraints is maximized.

Buffer blocks can be planned during post-floorplanning [32,30,39,93, 103] or floorplanning [26,59,60,85]. Planning buffer blocks during post-floorplanning is more efficient, but is often limited by the quality of a given floorplan since the location and size of the space for buffer insertion is fixed. Further, the dead spaces for buffer blocks are typically treated as unwanted cost during floorplanning, so they are often avoided or minimized. As a result, the size and location of a buffer block may not be suitable for later buffer insertion. Therefore, researchers also try to integrate buffer block planning into floorplanning to fully utilize *useful* dead spaces for performance optimization. This approach typically enjoys higher design flexibility, but inevitably incurs higher time complexity.

Cong et al. first consider buffer block planning during post-floorplanning in [32]; they derive feasible region formulae to determine where to insert buffers to meet timing constraints and propose a greedy algorithm to plan buffer blocks in a slicing floorplan. Sarkar et al. also consider routability and address the concept of independent feasible regions in [93]. Moreover, [32,93] expand channels to provide more buffers if necessary. Based on a network-flow formulation, Tang and Wong in [103] optimally plan as many buffers into buffer blocks as possible for all nets, each with at most one buffer. Given an existing buffer block plan, Dragan et al. in [39] perform buffering of global nets. They route the nets using available buffer blocks, such that required upper and lower bounds on buffer intervals and the wirelength upper bounds per connection are satisfied.

We describe the generic approach for buffer block planning at post-floorplanning as presented in [32]. First, it constructs a directed horizontal and a directed vertical constraint graphs for a given floorplan, denoted by G_H and G_V, respectively. Each vertex v in G_H corresponds to a vertical routing channel, and an edge $e = (v_1, v_2)$ represents a circuit block whose respective left and right boundaries are adjacent to the routing channels v_1 and v_2. The weight of a vertex v, $w(v)$, represents the corresponding channel width while the weight of an edge e, $w(e)$, represents the corresponding block width. The graph G_V can be constructed similarly. Applying a longest-path algorithm on G_H and G_V, we can obtain the respective width W_c and height H_c of the chip.

Then, we divide the dead spaces and routing channels into tiles to facilitate buffer block planning. For each tile, we compute its area slack with respect to the longest paths in G_H and G_V. For those dead spaces and routing channels not on the critical paths in the constraint graph G_H / G_V, we will have some positive area slacks in width/height. If there is still some net that needs buffer(s) to meet the timing constraint, we will pick a best tile for buffer insertion and then insert proper buffers into this tile. By a best tile, we mean that the tile with a largest positive area slack. If there is no tile with positive area slack, then any buffer insertion will need to shift some circuit block and thus increase the overall chip area. This shifting will make room for other tiles, so we will have some new positive-slack tiles. We choose the dead space or the routing channel that has the maximum buffer insertion demand and pick one tile in it. For the selected tile, we insert desired buffers into it. In case there is not sufficient space in the tile for buffer insertion, we will expand the corresponding routing channel to make room for the buffers. After the buffer insertion for the tile, we update the information of the constraint graphs, feasible regions, and the chip dimension and repeat the buffer insertion/clustering process until all buffers are placed.

More recently, researchers try to perform simultaneous buffer block planning and floorplanning to fully utilize *useful* dead spaces for performance optimization [26,59,60,85]. Jiang, et al. in [59,60] provides a generic paradigm along this direction. The work presents an algorithm that simultaneously considers floorplanning and buffer block planning. The method adopts simulated annealing to refine a floorplan so that buffers can be inserted more effectively. In each iteration, we construct a routing tree for each net and calculate the longest path from the source to the sink in each routing tree. Based on the aforementioned formulae presented in preceding sections, we obtain the number of buffers needed for the longest path, the optimal distance from the source terminal to each buffer, and the width of independent feasible region. After allocating buffers for all nets, we make buffer blocks as soft circuit blocks into the constraint graphs. These buffer blocks may occupy dead spaces or be inserted into routing channels. After all buffers for all nets are allocated, the area of each buffer block is determined as the bounding area of inserted buffers. We then reshape the floorplan by Lagrangian relaxation. Unlike the work for buffer block planning after floorplanning that generates buffer blocks *before* buffer assignment, in particular, this work generates buffer blocks *after* buffer assignment, and thus the area of buffer blocks can properly be controlled, especially for the buffer blocks in routing channels.

5.2 Floorplan and Power/Ground Network Co-Synthesis

5.2.1 Introduction

As technology advances, the metal width decreases while the global wire length increases. This trend makes the resistance of the power wire increase substantially. Further, the threshold voltage scales nonlinearly, raising the ratio of the threshold voltage to the supply voltage and making the voltage (IR) drop in the P/G network a serious challenge in modern SOC design [78]. Due to the IR-drop, supply voltage in logic may not be an ideal reference. This effect may weaken the driving capability of logic gates, reduce circuit performance, slow down slew rate (and thus increase power consumption), and lower noise margin [111].

Figure 51(a) shows a chip floorplan of four blocks and the P/G network. As shown in the figure, we refer to a pad feeding supply voltage into the chip as a *power pad*, the power line enclosing the floorplan as a *core ring*, a power line branching from a core ring into blocks inside as a *power trunk*, an intersection of a vertical and a horizontal power lines a *P/G node*, and a pin

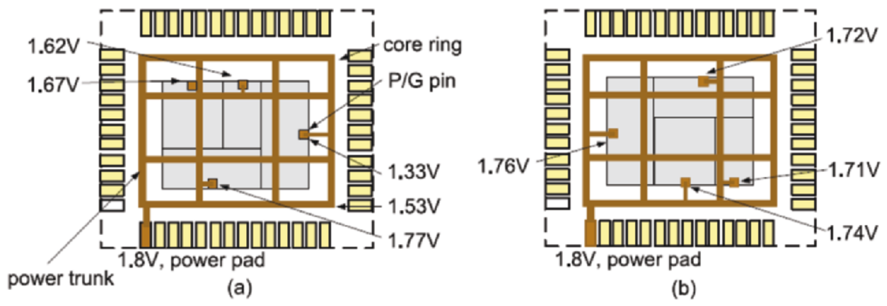


Figure 51. (a) An instance of floorplan and its P/G network structure. The worst-case voltage at the P/G pins is about 26% of the supply voltage. (b) A floorplan with smaller worst-case voltage drops. The worst-case voltage drop is about only 5%

in a block that absorbs current (connects to a core ring or a power trunk) as an *P/G pin*. To ensure correct and reliable logic operation, we shall minimize the IR drops from the power pad to the P/G pins in a P/G network. Figure 51(a) shows an instance of voltage drop in the power supply line, in which the voltage drops by almost 26% at the rightmost P/G pin. As [111] pointed out that 5% IR drop in supply voltage may slow down circuit performance by as much as 15% or more. Therefore, IR drop is a first-order effect and can no longer be ignored during the design process, and it is desired to consider the P/G network synthesis during early physical design (e.g., floorplanning) for reliable circuit operation.

The problem of P/G network synthesis has been studied extensively in the literature. An important problem of P/G network synthesis is to use the minimum amount of wiring area for a P/G network under the power integrity constraints such as IR drops and electromigration. There are two major tasks for the synthesis: (1) P/G network topology determination to plan the wiring topology of a P/G network [20] [97], etc. and (2) P/G wire sizing to meet the current density and reliability constraints [28] [104].

As the design complexity increases dramatically, it is necessary to handle the IR-drop problem earlier in the design cycle for better design convergence. Most existing commercial tools deal with the IR-drop problem at the post-layout stage when entire chip design is completed and detailed layout and current information are known. It is, however, often very difficult and computationally expensive to fix the P/G network synthesis at the post-layout stage. Therefore, researchers started to consider the P/G network analysis at an earlier design stage [37] [105] [111].

Dharchoudhury et al. proposed a design flow with different modes of power grid analysis incorporated between stages of the design flow [37]. The work shows that considering power integrity analysis at an earlier stage can significantly improve design convergence. Yim, Bae, and Kyung in [111] presented an early floorplan-based P/G network planning methodology.

Recently, Wu and Chang proposed a power integrity-driven design methodology of performing P/G network analysis after floorplanning [105].

It is very reasonable that [37], [105], and [111] can significantly improve design convergence. At the floorplanning stage, a prototype of the chip is determined and the power consumption for each block and the positions for blocks and P/G pins become available, making the P/G network analysis feasible at this stage. Furthermore, it is intrinsically more flexible to fix any power integrity problem at this stage than at the post-layout stage when most block positions and wiring are fixed. However, there is a significant difficulty in doing the early P/G network analysis: Traditional P/G network analysis methods are often very computationally expensive and are thus not feasible to be incorporated into the floorplanning design. To make the power integrity-driven design flow feasible, we need a very efficient, yet sufficiently accurate P/G network analysis method.

In this section, we introduce the method presented by Liu and Chang [82] for floorplan and P/G network *co-synthesis* based on an efficient, yet sufficiently accurate P/G network analysis scheme for the mesh P/G structure and the efficient B*-tree floorplan representation [17]. We introduce a P/G network aware method to reduce the floorplan solution space and thus speed up the co-synthesis, and then integrate the co-synthesis step into a commercial design flow to develop an effective power integrity (IR-drop) driven design flow for faster design convergence.

5.2.2 Problem definition

The problem of floorplan and P/G network co-synthesis is formulated as follows: Given a floorplan of m blocks, the number of power pads for the whole chip and the power consumption for each block, the objective is to obtain a feasible floorplan and simultaneously generate a corresponding P/G network that satisfies the power constraints. Before presenting the power integrity constraints, we introduce the notations for describing a P/G network used in [105]: Let $G = \{N, B\}$ be a P/G network with n nodes $N = \{1, 2, \ldots, n\}$ and b branches $B = \{1, 2, \ldots, b\}$. Each branch i in B connects two nodes: i_1 and i_2 with current flowing from i_1 to i_2. Let l_i and w_i be the length and width of branch i, respectively. Let $\hat{r}$ be the sheet resistivity (unit Ω per square), and V_i (I_i) be the voltage (current) at node i. Then the resistance r_i of branch i is $r_i = (V_{i_1} - V_{i_2})/I_i = \hat{r} l_i / w_i$. At the early stage power analysis, we need a fast analysis for the P/G network. For this reason, a sophisticated model for the P/G network is often too time-consuming and thus infeasible for the co-synthesis. In this section, we use the resistive model for P/G networks and the static current source model. We consider the power integrity constraints as follows:

- **The IR-drop constraints:**
  For every P/G pin i, its corresponding voltage V_i must satisfy the following constraints:
  $V_i \geq V_{min,k}$ for each power pin i of block k,
  $V_i \leq V_{max,k}$ for each ground pin i of block k,
  where $V_{min,k}(V_{max,k})$ is the minimum (maximum) voltage required at the injection point of a P/G network for block k.
- **The minimum width constraints:**
  The width of a P/G line must be greater than the minimum width allowed in the given technology. The constraint is given by

$$w_i = \frac{\hat{r} l_i I_i}{V_{i_1} - V_{i_2}} \geq w_{i,min}, \tag{30}$$

  where $w_{i,min}$ is the given constraint.
- **The electromigration constraints:**

  $|V_{i_1} - V_{i_2}| \leq \hat{r} l_i \sigma$ (i.e., $I_i / w_i \leq \sigma$), for each $i \in B$

  where σ is a constant for a particular routing layer with a fixed thickness.

5.2.3 The co-synthesis flow

In this section, we describe the floorplan and power/ground network co-synthesis flow proposed by Liu and Chang [82], which is illustrated in Figure 52. The netlist is the circuit generated in high-level synthesis. It partitions the circuit into hard blocks (hard macros) and soft blocks (groups of standard cells). The P/G network and floorplan co-synthesis generates a P/G network and a floorplan that satisfy all power integrity constraints.

With a feasible floorplan, it performs placement and routing which include detailed placement, P/G routing, clock tree synthesis, and detailed routing. Finally, the final P/G network is analyzed, and simulation is performed to check the correctness of the final design.

5.2.4 Floorplan and P/G network co-synthesis

In this section, we introduce the floorplan and P/G network co-synthesis algorithm. The floorplanning algorithm adopts the B*-tree floorplan

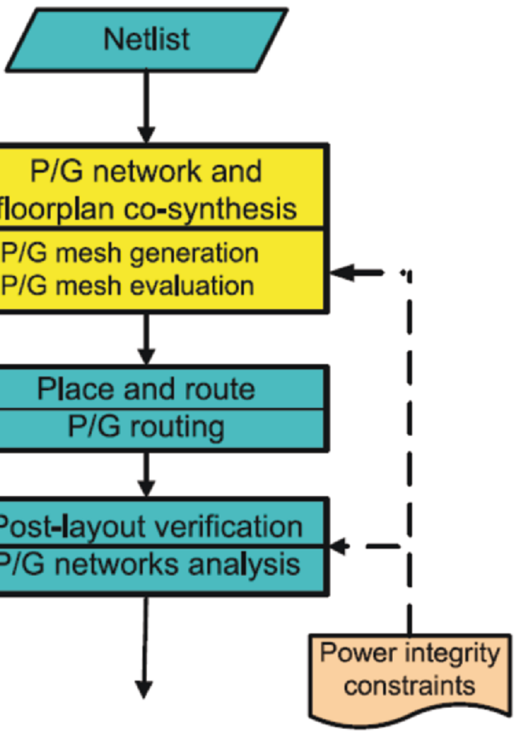


Figure 52. The floorplan and power/ground network co-synthesis flow

representation [17] and uses simulated annealing (SA). The SA algorithm requires a cost function to guide the optimization. To perform power integrity driven floorplanning, it adds a penalty for violating the power integrity constraints and the P/G mesh density cost in the cost function as follows:

$$\Psi = \alpha W + \beta A + \gamma \Phi + \omega \frac{A}{D_{pitch}^2}, \tag{31}$$
$$0 < \alpha, \beta, \gamma, \omega < 1, \ \ \alpha + \beta + \gamma + \omega = 1,$$

where W is the wirelength, A is the area, Φ is the penalty function of power integrity violations and D_{pitch} is the pitch of the P/G mesh which will be discussed in later sections, and α, β, γ, and ω are weighting parameters. The term A/D_{pitch}^2 is the density cost of the P/G mesh which affects the routing resource. The cost function is calculated after packing a B*-tree to obtain a corresponding floorplan. To obtain the penalty function of power integrity violations, we first generate a P/G mesh for the floorplan and then evaluate the P/G mesh. In the following sections, we discuss the P/G mesh generation and the evaluation method.

A: P/G Mesh Generation

In order to evaluate the performance of the actual P/G network of a floorplan at the floorplanning stage, it generates a conceptual P/G network for the floorplan. We use the mesh structure for the P/G network, since it is widely used in modern VLSI chips to reduce the IR-drop effects. By specifying the pitch of the power lines, it can determine the dimension of the P/G mesh. A uniform mesh can then be generated easily by evenly distributing the power lines. Figure 53(a) shows a uniform mesh.

The pitch D_{pitch} of the P/G mesh is determined during the SA process and depends on the average value of the P/G network penalty function Φ. We will detail the determination of D_{pitch} later.

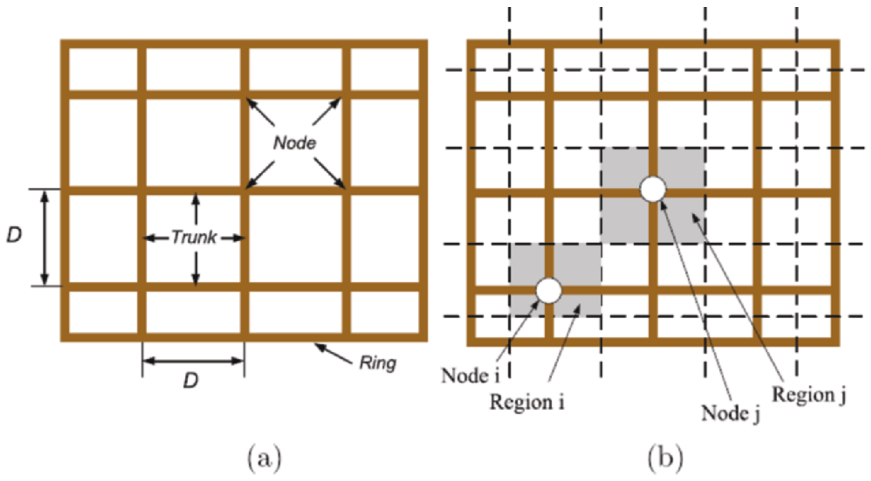


Figure 53. (a) A uniform P/G mesh. (b) A floorplan with a P/G mesh divided into regions

The complexity of the P/G mesh analysis mainly depends on the number of nodes of the mesh. To reduce the complexity, it makes a reasonable approximation by attaching all current sources to the intersection nodes of the vertical and horizontal power lines. That is, every P/G pin is connected to its nearest node with a power strap, and the length of the strap is the Manhattan distance between the P/G pin and the node. For convenience, it divides the floorplan into n regions, where n is the number of the nodes. The divided floorplan is illustrated in Figure 53(b). The border line of two regions is the center line between the two nodes such that the node is the nearest one for any point in the region.

B: Macro Current Source Modelling

In [74], it is shown that the result of static P/G analysis can be an upper bound for that of dynamic analysis by using the peak current. Therefore, they consider static analysis using constant current sources with the maximum current. We introduce how to estimate the maximum current consumption of hard and soft blocks. For hard blocks, it connects a P/G pin to the corresponding (center) node of the region where the pin is located, and the pin absorbs the estimated maximum current consumed by the pin, which is obtained by the pattern-based power simulation. At the floorplanning stage, it does not have the exact placement of the standard cells in the soft block. For soft blocks, therefore, its current model is based on the worst-case scenario. It uses the *maximum possible current function*, $I_{max}()$, to determine the current assigned to the nodes. The definition of $I_{max}(A_r, k)$, the maximum possible current in the specified region of the soft block k with size A_r, is as follows:

$$I_{max}(A_r, k) = \max_{S(A_r,k)} \left(\sum_{\forall i \in S_n} I_c(i) \right), \tag{32}$$

where $S(A_r, k)$ is the set of sets of standard cells in the soft block k, such that for each set $S_n \in S(A_r, k)$, $\sum_{\forall i \in S_n} A_r(i) \le A_r$ ($A_r(i)$ is the area of the standard cell i) and $I_c(i)$ is the maximum estimated current drawn by the cell i. The problem of solving $I_{max}()$ can be formulated as a 0-1 knapsack problem [36]: The area is the total weight that one can carry, the area of a cell is the weight of an item, and the current drawn by the cell is the value of the item. The goal is to take as valuable a load as possible while the total weight of items does not exceed a given total weight constraint. Since the 0-1 knapsack problem is NP-complete [36], it is computationally expensive to solve the problem exactly. Therefore, they resort to an approximation by assuming that each standard cell can be divided freely. Then the maximum possible current can be approximated efficiently in linear time using the fractional knapsack algorithm [36]. As Figure 54 illustrates, for the soft block k overlapping with the region n, $I_{max}(A_{ov}(n,k), k)$ amount of current is assigned to the node n, where $A_{ov}(n,k)$ is the amount of the area k overlapping with n. Taking the node n as an example, its region (region n) contains two pins of the block A and three pins of the block B. Assume that the gray area is equal to the total area of 10 cells. Thus, there are 10 cells

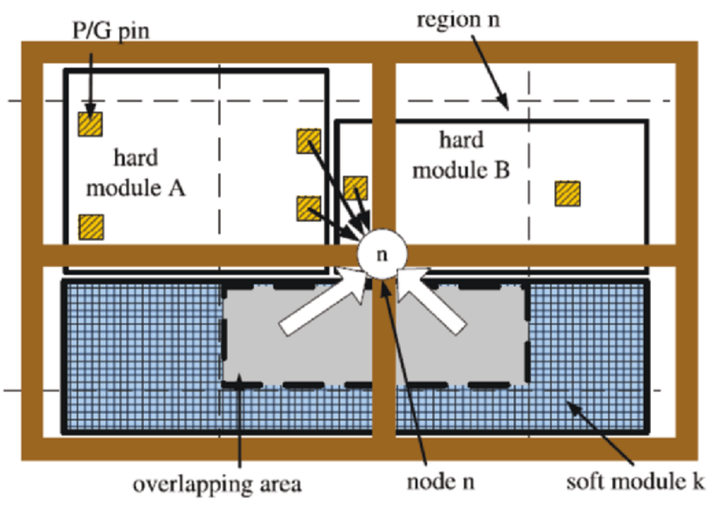


Figure 54. An example of the P/G analysis. The dashed lines denote the boundaries of the regions, and the gray area denotes the overlap of the soft block k and the region n. Each pin in the block A absorbs $0.3A$ current and each pin in the block B absorbs $0.5A$ current. The soft block k contains 30 standard cells of the same size. The largest current-consuming cell draws $30mA$ current, the second one draws $29mA$ current, and so on. Therefore the smallest cell draws $1mA$ current

with from 30 mA to 21 mA current of the block k being attached to node n. Therefore, the current source attached to the node n consumes $0.3 \times 2 + 0.5 + (0.03 + 0.021) \times 10/2 = 1.355A$ current.

Since the external voltage supply is typically connected to the ring, all voltage sources are assigned to the nodes on the ring. Then, the number of voltage supplies and the maximum current per supply node depend on the power budget of the design.

C: P/G Networks Analysis

After the P/G network is generated, it analyzes the P/G mesh with the floorplan. Traditional analysis for a complete and accurate P/G network is very computationally expensive and unaffordable for integrating with floorplanning. The objective for floorplan and P/G network co-synthesis is to derive an efficient scheme for the P/G network analysis based on the technology information available at the floorplanning stage. They apply the resistive P/G network model [81] and use the maximum current drawn by the blocks for static P/G network analysis. As the P/G mesh example shown in Figure 55, the chip is composed of four blocks. The P/G wires are modelled as resistors. A P/G pin in a hard block is modelled as a current source.

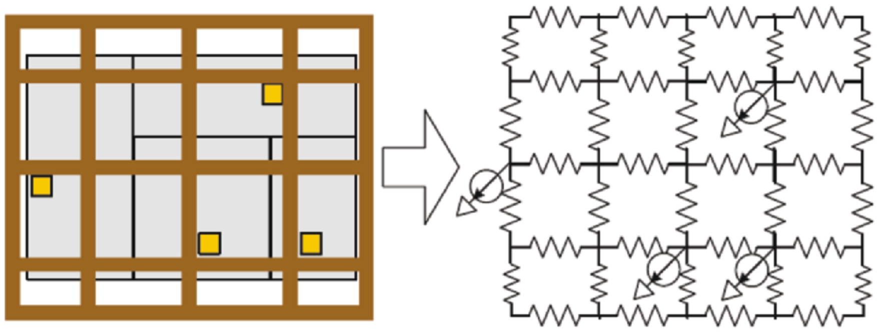


Figure 55. A global power mesh and its equivalent circuit model

The static analysis of a P/G network is formulated as follows [81]:

$$\mathbf{Gx} = \mathbf{i}, \tag{33}$$

where **G** is the conductance matrix for the resistor, **x** is the vector of node voltages, and **i** is the vector of current loads. The dimensions of **i** and **x** are equal to the number of nodes in the P/G network, and **G** is a sparse positive definite matrix for a general resistor network.

They solve Equation (36) efficiently by using an iterative method for the sparse matrix such as the preconditioned conjugated gradient method and/or other Krylov subspace methods [47]. The time complexity of solving the equation is $O(n)$, where n is the number of the nodes in the mesh. As mentioned in the preceding section, we reduce the number of nodes by an approximation presented in the preceding subsection. Thus the number n is within a tractable range.

Once the voltage of each node is obtained, they estimate the voltage at each P/G pin based on the voltage of the closest (connected) node and the distance of the P/G pin. For a hard block, the voltage of a P/G pin is estimated by the voltage of the closest node minus the largest possible voltage drop over the strap connecting the node and the pin. For a P/G pin j and its corresponding node i, the estimation is given by

$$V_j = V_i - I_j \max\left(\hat{r}_h \frac{Dx_{ij}}{w_{hstrap}}, \hat{r}_v \frac{Dy_{ij}}{w_{vstrap}}\right), \tag{34}$$

where $\hat{r}_v$ and $\hat{r}_h$ are the respective sheet resistivity of the vertical and horizontal metal layers, w_{hstrap} and w_{vstrap} are the widths of the respective vertical and horizontal straps, Dx_{ij} and Dy_{ij} are the respective vertical and horizontal distances between pin j and node i. For example, the left pin of the block B in Figure 54 is estimated by the voltage of the node n, which is 1.78 V. The current consumption of the pin is $0.5A$, the horizontal sheet resistivity is $5m\Omega/$ unit square, the vertical sheet resistivity is $4m\Omega/$ unit square, the respective vertical and horizontal distances from the pin to the node n are $5\mu m$ and $3\mu m$, and the width of a strap is $1\mu m$. The estimated voltage of the pin is $1.78 - 0.5 \times \max(0.005 \times \frac{5}{1}, 0.004 \times \frac{3}{1}) = 1.77V$. For a soft block, they use the distance between the center of the overlapping area and the node as the length of the strap. The voltage is estimated by the lowest supply voltage of the soft block k (a block may be attached to more than one node) as follows:

$$V_k = \min_{S_{ov}} \left(V_i - I_{k,i} \max \left(\hat{r}_h \frac{Dx_{ik}}{w_{hstrap}}, \hat{r}_v \frac{Dy_{ik}}{w_{vstrap}} \right) \right), \tag{35}$$

where S_{ov} is the set of nodes responsible for the soft block k, $I_{k,i}$ is the current supplied by node i, and Dx_{ik} and Dy_{ik} are the respective horizontal and vertical distances between the node i and the center of the overlapped area. Again let us take the node n in Figure 54 as an example. The vertical and horizontal distances between the center of the gray area and the node n are $6\mu m$ and 0μ, respectively. The estimated voltage of the block k with respect to the node n is $1.78 - ((0.03 + 0.021) \times \frac{10}{2}) \times 0.004 \times \frac{6}{1} = 1.774V$. Assume that this is the lowest voltage among all the estimated voltages calculated from all regions overlapped with the block k. Thus, the estimated voltage of the block k is $1.774V$. Now it can verify the power integrity constraints. The IR-drop constraints is verified by checking the IR drop of each P/G pin, and the electromigration constraints can be verified by checking the current flowing through every branch of the P/G mesh.

Now we can derive Φ, the penalty function of power integrity violations. The function Φ is given as follows:

$$\Phi = \theta \frac{|B_{em}|}{|B|} + (1-\theta) \frac{\sum_{\forall pv_i \in P_v} v_{pv_i}}{\sum_{\forall p_i \in P} V_{lim,p_i}}, \quad 0 < \theta < 1, \tag{36}$$

where θ is a weighting parameter, B_{em} is the set of branches violating electromigration constraints, B is the total branches of the P/G mesh, v_{pv_i}

is the amount of the violation at the pin pv_i, P is the set of all P/G pins, P_v is the set of violating P/G pins, and V_{lim,p_i} is the IR-drop constraint of the P/G pin p_i ($Vdd - V_{min,p_i}$ for a power pin and V_{max,p_i} for a ground pin). The first part of the right-hand side denotes the ratio of branches violating the electromigration constraints over total branches, and the second part denotes the ratio of the amount of IR-drop violation over the total amount of possible violations. The denominators are for the penalty normalization.

D: P/G Network Co-synthesis Heuristic

According to their experience, if the pitch is carefully chosen, the algorithm can find desired floorplans with very few constraint violations at high temperatures and continue to optimize wirelength and area at lower temperatures, leading to high-quality floorplan solutions. Note that IR drop and the current per branch decrease as the density of the mesh increases; therefore, the P/G violation penalty Φ can be reduced by increasing the density of the mesh. Since the density of a P/G mesh is proportional to A/D_{pitch}^2, we can control D_{pitch} instead of the density for convenience. By controlling D_{pitch} during the SA process, it can obtain desired floorplan solutions. It updates the P/G mesh pitch D_{pitch} at each temperature by multiplying k_i, which is defined as follows:

$$k_i = \frac{\hat{\Phi}}{\Phi_{avg,i}}, \tag{37}$$

where $\Phi_{avg,i}$ is the average of Φ at the temperature of the ith iteration during the SA process, and $\hat{\Phi}$ is expected average of Φ, which a user-specified parameter. The floorplans generated at the same temperature form a solution sub-space. Specifying $\hat{\Phi}$, it can control the average Φ of the solution sub-space and statistically control the proportion of the feasible solutions in the solution sub-space.

E: Feasible B*-trees with Power Mesh Constraints

They study the properties of the B*-tree with the P/G network considerations and develop techniques to reduce the solution space to speed up the search for desired floorplans. Finding the best positions of blocks to optimize the P/G mesh is a very complex problem. Their idea is motivated by the linear circuit theory: the IR drop of a P/G pin is proportional to the effective resistance between the P/G pin and the power pad. Therefore, the closer the P/G pin is placed to the power pad, the smaller IR drop we can get. Based on this fact, it places the blocks which consume larger current near the boundary of the floorplan, and then place power pads close to them. To implement this idea, they sort the blocks by their power consumption and cluster the leading blocks, which are called *power-hungry blocks* to form groups. In their implementation, they chose 10% of total blocks to be power-hungry blocks. The size of a group depends on the total size of the member blocks, which is a user specified parameter. Note that each group should contain at least one block. These groups are referred to as *power-hungry groups*. Each power-hungry group is assigned with a power pad and the number of the groups equals the number of available power pads. In order to reduce the IR drops of power-hungry groups, it prefers to place the blocks in the power-hungry groups along the boundary of the floorplan And it will place each pad next to a power-hungry group.

There are two goals for the floorplan and power/ground network co-synthesis: (1) place power-hungry groups along the chip boundary, and (2) maintain all the power-hungry blocks in power-hungry groups, which can be accomplished by careful perturbations and will be discussed later. For the first goal, we should identify the boundary blocks of the floorplan. Now we explore the feasibility conditions of the B*-tree to search for desired floorplan solutions. Let the *boundary ring* Υ_F (Υ_T) of the floorplan F (the B*-tree T) be the ordered list of the boundary blocks in F (T) (say, in the counter-clockwise sequence starting from the block at the bottom-left corner). For example, $\Upsilon_F = < m_0, m_1, m_2, m_5, m_6, m_9, m_8, m_7, m_3 >$ ($\Upsilon_T = < n_0, n_1, n_2, n_5, n_6, n_9, n_8, n_7, n_3 >$) in the floorplan F (the B*-tree T) of Figure 56. Notice that by the name "ring", the succeeding element of the last element in the "list" can be treated as the first element of the list. For the example of Figure 56, m_0 (n_0) is the succeeding element of m_3 (n_3). We shall make all blocks of the power groups belong to the blocks in the boundary ring such that the blocks of the same power group are placed in the order according to the boundary "ring."

Extending the findings in [79] by Lin et al., they identify the blocks in the boundary ring based on the feasibility conditions of B*-trees for boundary blocks. Let the root of the B*-tree T be r, the DFS order of the tree traversal on the leftmost and the bottom-left branches of T be L_T, and the DFS order of the tree traversal on the rightmost and the bottom-right branches of T be R_T. Let the reverse of a sequence L be L^r. Then, we have $\Upsilon_T = L_T \oplus R_T^r$. Here, "$\oplus$" denotes the concatenation operation of two lists.

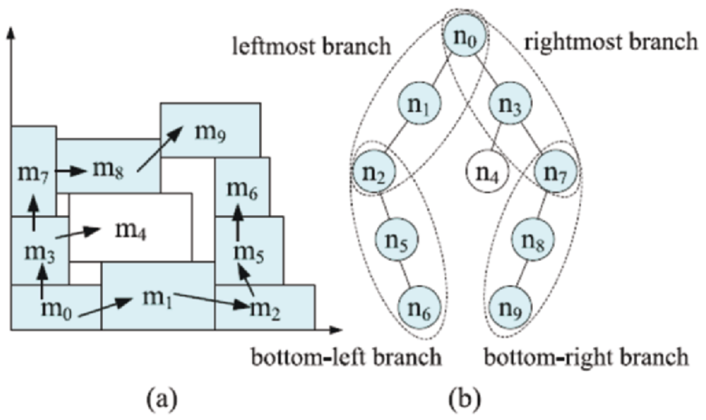


Figure 56. Boundary blocks and their corresponding B*-tree branches

Theorem 6 *(Boundary Ring)* $\Upsilon_T = L_T \oplus R_T^r$.

According to Theorem 6, we shall make the nodes corresponding to the blocks of a power-hungry group in the boundary ring Υ_T. In other words, it prefers to make those nodes a sublist of the ring Υ_T during the perturbation in simulated annealing. As shown in the example of Figure 57, the power group $\{m0, m1, m3\}$ ($\{m6, m8, m9\}$) is placed on the left and the bottom (the right and the top) boundaries close to the bottom-left (top-right) corner, and they are adjacent blocks in the ring Υ_F. A floorplan is said to be

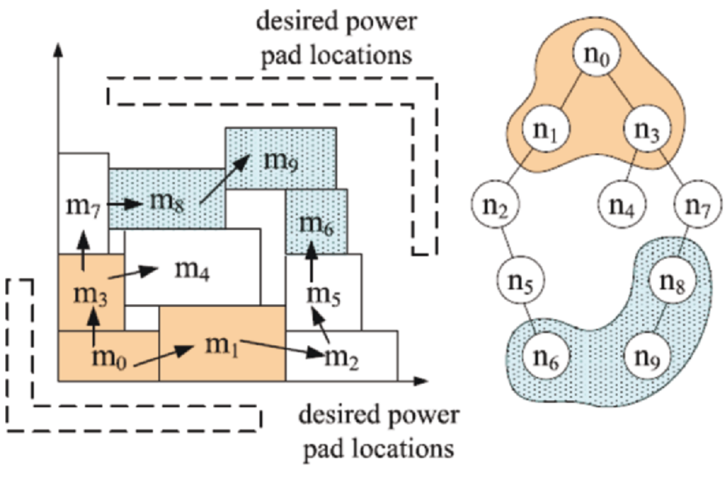


Figure 57. An example of a power-feasible floorplan with two power groups: $\{m6, m8, m9\}$ and $\{m0, m1, m3\}$. The desired power pad locations are encircled by the dashed lines

power-feasible if the power-hungry blocks in each power-hungry group are blocks in the desired locations of the boundary ring. Therefore, it is desirable to keep a power-feasible floorplan during solution perturbation to achieve the second goal of the co-synthesis.
While perturbing the tree, the power-feasibility of the B*-tree is maintained. The operations to perturb a B*-tree [17] with the IR-drop consideration are listed as follows:

- Op1: Rotate a block.
- Op2: Swap two blocks in the power-hungry groups or not in any power-hungry group.
- Op3: Move a block to another place that maintains power-feasibility.

Op1 only exchanges the width and height of a block without changing the B*-tree topology while Op2 and Op3 do. Therefore, in order to maintain the power-feasibility, it only swaps two blocks in power-hungry groups or not in any power-hungry group for Op2, and move a block to another place that maintains power-feasibility for Op3. Otherwise, it might need to transform the B*-tree to maintain the power-feasibility.

F: The Co-Synthesis Algorithm

Figure 58 summarizes the floorplaning algorithm. Given inputs of the block information, initial P/G pitch D_{pitch}, and power integrity constraints, it starts with the simulated annealing process (see lines 2–24). At the beginning of simulated annealing, it randomly explores the solution space to get an average cost to normalize each objective in the cost function (line 3). Then it gets an initial solution and an initial temperature (lines 4–6) and launches the simulated annealing process. At each temperature, it anneals for N times, where N is a number proportional to the number of blocks (line 8). After each perturbation (line 9), it computes the coordinates of all blocks and constructs a P/G mesh (lines 10–11). Then it calculates the voltage of each node of the mesh by solving Equation (33) using their linear solver and estimates the IR drop of each P/G pin by Equations (34) and (35) (lines 12–13). Then it calculates the P/G mesh penalty function Φ and accumulates it for the average bookkeeping (line 14). Next it updates the cost function by Equation (31) and checks if the floorplan is accepted with the probability $e^{\frac{-\Delta\Psi}{T}}$ (lines 15–20). If the current floorplan S has a lower cost than the best floorplan S_{best} found so far, S is chosen as the best floorplan (line 20). Next, it calculates $\Phi_{avg,i}$ and k_i, and then updates the mesh pitch D_{pitch} by $k_i D_{pitch}$ to co-synthesize the P/G mesh (lines 21–22).

```
Algorithm: Power Integrity Aware Floorplanning
1  Read initial settings: block information,
   initial pitch $D_{pitch}$, power integrity constraints,
   and power consumption data;
2  do
3      Get an average cost to normalize the cost;
4      Get an initial power-feasible floorplan $S$;
5      $S_{best} \leftarrow S$;
6      Get a temperature $T > 0$;
7      Start with the simulated annealing process;
8      for 1 to $N$
9          Perturb the floorplan and maintain power feasibility;
10         Pack the floorplan;
11         Construct a P/G mesh;
12         Calculate the voltage of each node of the mesh;
13         Estimate the IR drop of each P/G pin;
14         Calculate and accumulate $\Phi$;
15         Calculate $\Psi$;
16         $\triangle\Psi \leftarrow \Psi(S') - \Psi(S)$;
17         if $\triangle\Psi \leq 0$ then $S \leftarrow S'$;
18         else if $\triangle\Psi > 0$ then
19                 $S \leftarrow S'$ with probability $e^{\frac{-\triangle\Psi}{T}}$;
20         if $\Psi(S) < \Psi(S_{best})$ then $S_{best} \leftarrow S$;
21     Calculate $\Phi_{avg,i}$ and $k_i$;
22     $D_{pitch} \leftarrow k_i D_{pitch}$;
23     $T \leftarrow rT$;
24 while not converged or not cooled down;
25 return $S_{best}$;
```

Figure 58. The P/G network and floorplan co-synthesis algorithm

At the end of the SA loop, it decreases the temperature T by multiplying a constant r (line 23).

6. CONCLUSION

We have introduced the state-of-the-art design algorithms and frameworks for the three major physical design steps: floorplanning, placement, and routing considering the impacts arising from modern SOC designs. With the breathtaking speed in which the design complexity increases, hierarchical and multilevel frameworks are essential to handle the very large-scale SOC design and optimization. The traditional hierarchical framework can scale very well to large-scale design, but it may lose the global view for circuit optimization because of its lack of interactions among subregions after partitioning. Special treatments are needed to deal with the optimization of global circuit effects. Two types of multilevel frameworks, the Λ- and V-shaped frameworks, have recently been studied in the literature. Both are based on two-stage techniques. The Λ-shaped framework adopts bottom-up coarsening followed by top-down uncoarsening, while the V-shaped framework proceeds with top-down uncoarsening and followed by bottom-up coarsening. Since the V-shaped framework processes global circuit regions first, it tends to obtain better solutions for those with global effects such as wirelength, timing, and crosstalk. In contrast, the Λ-shaped framework tends to achieve better solutions for local effects such as area optimization.

We have also introduced the interconnect-driven and signal/power integrity aware design methodologies for modern SOC designs to improve design convergence. When the complexity for interconnect and power/ground network designs grow drastically, it is often too late to perform aggressive interconnect and power/ground network optimization during or after routing since most silicon and routing resources are occupied. Therefore, it is desirable to optimize interconnect and power/ground network earlier at the flooprlanning/post-floorplanning stage. With the complexity continuing to grow for SOC design, we expect that more and more circuit effects will need to be handled earlier for fast design convergence.

With the continued increase of on-chip packing density and the continued shrinking of component feature sizes due to the nanometer IC technologies, some other issues such as thermal, reliability (antenna effect, electrostatic discharge, electro-migration, etc.), manufacturability (optical proximity effect, phase-shift mask, metal fill, etc.), and yield (redundant via, process variation, etc.) will soon become first-order effects for SOC design, as comparably important as the traditional design metrics—timing, power, signal/power integrity, and area. These effects have imposed tremendous challenges and opened many research opportunities to modern physical design.

REFERENCES

[1] L. C. Abel. On the ordering of connections for automatic wire routing. IEEE Transations on Computers, pages 1227-1233, November 1972.

[2] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov. Unification of partitioning, placement and floorplanning. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 550-557, 2004.

[3] S. N. Adya and I. L. Markov. Consistent placement of macroblock using floorplanning and standard-cell placement. In Proceedings of ACM International Symposium on Physical Design, pages 12-17, 2002.

[4] S. N. Adya and I. L. Markov. Combinatorial techniques for mixed-size placement. ACM Transactions on Design Automation of Electronics Systems, 10(1):58-90, January 2005.

[5] S. N. Adya, I. L. Markov, and P. G. Villarrubia. On whitespace in mixed-size placement and physical synthesis. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 311-318, 2003.

[6] S. B. Akers. A modification of Lee's path connection algorithm. IEEE Transations on Electronic Computers, pages 97-98, February 1967.

[7] C. J. Alpert and A. Devgan. Wire segmenting for improved buffer insertion. In Proceedings of ACM/IEEE Design Automation Conference, pages 588-593, June 1997.

[8] K. Arrow, L. Huriwicz, and H. Uzawa. Studies in Nonlinear Programming. Stanford University Press, Stanford, Calif, 1958.

[9] H. B. Bakoglu. Circuits, Interconnections, and Packaging for VLSI. Addison-Wesley, 1990.

[10] G. Blakiewicz, M. Jeske, M. Chrzanowska-Jeske, and J. S. Zhang. Substrate noise modeling in early floorplanning of mixed-signal SOCs. In Proceedings of IEEE/ACM Asia South Pacific Design Automation Conference, pages 819-823, 2005.

[11] M. Burstein and R. Pelavin. Hierarchical wire routing. IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, CAD-2(4):223-234, October 1983.

[12] M. Burstein and M. N. Youssef. Timing influenced layout design. In Proceedings of ACM/IEEE Design Automation Conference, pages 124-130, 1985.

[13] A. Caldwell, A. Kahng, and I. Markov. Can recursive bisection alone produce routable placement? In Proceedings of ACM/IEEE Design Automation Conference, 2000.

[14] H. H. Chan, S. N. Adya, and I. L. Markov. Are floorplan representations important in digital design? In Proceedings of ACM International Symposium on Physical Design, pages 129-136, 2005.

[15] T. Chan, J. Cong, and K. Sze. Multilevel generalized force-directed method for circuit placement. In Proceedings of ACM International Symposium on Physical Design, pages 185-192, 2005.

[16] C.-C. Chang, J. Cong, and X. Yuan. Multi-level placement for large-scale mixed-size ic designs. In Proceedings of IEEE/ACM Asia South Pacific Design Automation Conference, pages 325-330, 2003.

[17] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu. B*-trees: A new representation for non-slicing floorplans. In Proceedings of ACM/IEEE Design Automation Conference, pages 458-463, 2000.

[18] Y.-W. Chang and S.-P. Lin. MR: A new framework for multilevel full-chip routing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 23(5):793-800, May 2004.

[19] Y.-W. Chang, K. Zhu, and D.-F. Wong. Timing-driven routing for symmetrical-array-based fpgas. ACM Transactions on Design Automation of Electronics Systems, 5(3):433-450, July 2000.

[20] H. Chen, C.-K. Cheng, A. B. Kahng, M. Mori, and Q. Wang. Optimal planning for mesh-based power distribution. In Proceedings of IEEE/ACM Asia South Pacific Design Automation Conference, pages 444-449, 2004.

[21] T.-C. Chen and Y.-W. Chang. IMF: Interconnect-driven multilevel floorplanning for large-scale building-module designs. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, 2005.

[22] T.-C. Chen and Y.-W. Chang. Modern floorplanning based on fast simulated annealing. In Proceedings of ACM International Symposium on Physical Design, pages 104-112, 2005.

[23] T.-C. Chen and Y.-W. Chang. Multilevel gridless routing considering optical proximity correction. In Proceedings of IEEE/ACM Asia South Pacific Design Automation Conference, pages 1160-1163, January 2005.

[24] T.-C. Chen, Y.-W. Chang, and S.-C. Lin. A novel framework for multilevel full-chip gridless routing. In Proceedings of IEEE/ACM Asia South Pacific Design Automation Conference, January 2006.

[25] T.-C. Chen, T.-C. Hsu, Z.-W. Jiang, and Y.-W. Chang. NTUplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs. In Proceedings of ACM International Symposium on Physical Design, pages 236-238, 2005.

[26] Y.-H. Cheng and Y.-W. Chang. Integrating buffer planning with floorplanning for simultaneous multi-objective optimization. In aspdac, pages 624-627, Piscataway, NJ, USA, 2004. IEEE Press.

[27] M. Cho, H. Shin, and D. Z. Pan. Fast substrate noise-aware floorplanning with preference directed graph for mixed-signal socs. In Proceedings of IEEE/ACM Asia South Pacific Design Automation Conference, January 2006.

[28] S. Chowdhury. Optimum design of reliable ic power networks having general graph topologies. In Proceedings of ACM/IEEE Design Automation Conference, pages 787-790, 1989.

[29] J. Cong, J. Fang, M. Xie, and Y. Zhang. MARS-a multilevel full-chip gridless routing system. IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, 24(3):382-394, March 2005.

[30] J. Cong, J. Fang, and Y. Zhang. Multilevel approach to full-chip gridless routing. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 396-403, 2001.

[31] J. Cong, L. He, K.-Y. Khoo, C.-K. Koh, and Z. Pan. Interconnect design for deep submicron ICs. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 478-485, 1997.

[32] J. Cong, T. Kong, and D. Z. Pan. Buffer Block Planning for Interconnect-Driven Floorplanning. Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 358-363, 1999.

[33] J. Cong and P. H. Madden. Performance driven global routing for standard cell design. In Proceedings of ACM International Symposium on Physical Design, pages 73-80, April 1997.

[34] J. Cong, M. Romesis, and J. R. Shinnerl. Fast floorplanning by look-ahead enabled recursive bipartitioning. In Proceedings of IEEE/ACM Asia South Pacific Design Automation Conference, 2005.

[35] J. Cong, M. Xie, and Y. Zhang. An enhanced multilevel routing system. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 51-58, November 2002.

[36] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms. The MIT Press/McGraw-Hill Book Company, 2nd edition, 2001.

[37] A. Dharchoudhury, R. Panda, D. Blaauw, R. Vaidyanathan, B. Tutuianu, and D. Bearden. Design and analysis of power distribution networks in powerpc microprocessors. In Proceedings of ACM/IEEE Design Automation Conference, pages 738-743, 1998.

[38] K. Doll, F. M. Johannes, and K. J. Antreich. Iterative placement improvement by network flow methods. IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, 13(10):1189-1200, 1994.

[39] F. F. Dragan, A. B. Kahng, I. Mandoiu, S. Muddu, and A. Zelikovsky. Provably good global buffering by multi-terminal multicommodity flow approximation. In Proceedings of IEEE/ACM Asia South Pacific Design Automation Conference, pages 120-125, New York, NY, USA, 2001. ACM Press.

[40] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel. Chip layout optimization using critical path weighting. In Proceedings of ACM/IEEE Design Automation Conference, pages 133-136, 1984.

[41] A. E. Dunlop and B. Kernighan. A procedure for placement of standard-cell VLSI circuits. IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, CAD-4, January 1985.

[42] H. Eisenmann and F. M. Johannes. Generic global placement and floorplanning. In Proceedings of ACM/IEEE Design Automation Conference, pages 269-274, 1998.

[43] W. C. Elmore. The transient response of damped linear networks with particular regard to wide-band amplifiers. Journal of Applied Physics, 19(1):55-63, Jan. 1948.

[44] W. N. et al. Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer. In US Patent 6301693, October 2001.

[45] L. C. Evans. Partial Diferential Equations. American Mathematical Society, Providence, 2002.

[46] C. M. Fidducia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In Proceedings of ACM/IEEE Design Automation Conference, pages 175-181, 1982.

[47] G. H. Golub and V. L. C. F. Matrix Computations. Johns Hopkins University Press, 1996.

[48] P.-N. Guo, C.-K. Cheng, and T. Yoshimura. An O-tree representation of non-slicing floorplan and its applications. In Proceedings of ACM/IEEE Design Automation Conference, pages 268-273, 1999.

[49] F. O. Hadlock. A shortest path algorithm for grid graphs. Networks, pages 323-334, 1977.

[50] T. Hamada, C. K. Cheng, and P. M. Chau. Prime: A placement tool using a piece wise linear resistive network approach. In Proceedings of ACM/IEEE Design Automation Conference, pages 531-536, 1993.

[51] M. Hayashi and S. Tsukiyama. A hybrid hierarchical approach for multi-layer global routing. In Proceedings of European Design and Test Conference, pages 492-496, 1995.

[52] D. Hightower. A solution to line routing problems on the continuous plane. In Proceedings of Design Automation Workshop, pages 1-24, 1969.

[53] T.-Y. Ho, C.-F. Chang, Y.-W. Chang, and S.-J. Chen. Multilevel full-chip routing for the x-based architecture. In Proceedings of ACM/IEEE Design Automation Conference, pages 597-602, June 2005.

[54] T.-Y. Ho, Y.-W. Chang, and S.-J. Chen. Multilevel routing with antenna avoidance. In Proceedings of ACM International Symposium on Physical Design, pages 34-40, April 2004.

[55] T.-Y. Ho, Y.-W. Chang, S.-J. Chen, and D.-T. Lee. A fast crosstalk- and performance-driven multilevel routing system. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 382-387, November 2003.

[56] T.-Y. Ho, Y.-W. Chang, S.-J. Chen, and D.-T. Lee. Crosstalk- and performance-driven multilevel full-chip routing. IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, 24(6):869-878, 2005.

[57] T. C. Hu and M.-T. Shing. A decomposition algorithm for circuit routing. In VLSI Circuit Layout: Theory and Design, pages 144-152. IEEE Press, New York, NY, 1985.

[58] M. Jackson and E. S. Kuh. Performance-driven placement of cell based IC's. In Proceedings of ACM/IEEE Design Automation Conference, pages 370-375, 1989.

[59] H.-R. Jiang, Y.-W. Chang, J.-Y. Jou, and K.-Y. Chao. Simultaneous Floorplanning and Buffer Block Planning. Proceedings of IEEE/ACM Asia South Pacific Design Automation Conference, pages 431-434, 2003.

[60] H.-R. Jiang, Y.-W. Chang, J.-Y. Jou, and K.-Y. Chao. Simultaneous Floorplan and Buffer Block Optimization. IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, 23(5):694-703, 2004.

[61] Z.-W. Jiang, T.-C. Chen, T.-C. Hsu, H.-C. Hsu, and Y.-W. Chang. NTUplace2: a hybrid placement tool using partitioning and analytical techniques. In Proceedings of ACM International Symposium on Physical Design.

[62] A. B. Kahng, I. Markov, and S. Reda. On legalization of row-based placements. In Proceedings of ACM Great Lakes Symposium on VLSI, pages 214-219, 2004.

[63] A. B. Kahng and S. Reda. Placement feedback: a concept and method for better min-cut placements. In Proceedings of ACM/IEEE Design Automation Conference, pages 357-362, 2004.

[64] A. B. Kahng, S. Reda, and Q. Wang. Architecture and details of a high quality, large-scale analytical placer. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 890-897, 2005.

[65] A. B. Kahng, P. Tucker, and A. Zelikovsky. Optimization of linear placements for wirelength minimization with free sites. In Proceedings of IEEE/ACM Asia South Pacific Design Automation Conference, pages 241-244, 1999.

[66] A. B. Kahng and Q. Wang. An analytic placer for mixed-size placement and timing-driven placement. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 565-572, 2004.

[67] A. B. Kahng and Q. Wang. An analytic placer for mixed-size placement and timing-driven placement. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 565-572, 2004.

[68] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. In Proceedings of ACM/IEEE Design Automation Conference, page 526-529, 1997.

[69] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh. Pattern routing: Use and theory for increasing predictability and avoiding coupling. In IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, pages 777-790, November 2002.

[70] A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C.-K. Koh, and P. H. Madden. Recursive bisection based mixed block placement. In Proceedings of ACM International Symposium on Physical Design, pages 84-89, 2004.

[71] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598):671-680, 1983.

[72] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich. Gordian: Vlsi placement by quadratic programming and slicing optimization. IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, 10.

[73] T. Kong. A novel net weighting algorithm for timing-driven placement. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 172-176, 2002.

[74] D. Kouroussis and F. N. Najm. A static pattern-independent technique for power grid voltage integrity verification. In Proceedings of ACM/IEEE Design Automation Conference, pages 99-104, 2003.

[75] C. Y. Lee. An algorithm for path connection and its application. IRE Transactions on Electronic Computer, EC-10, pages 346-365, 1961.

[76] H.-C. Lee, J.-M. Hsu, Y.-W. Chang, and H. Yang. Multilevel floorplanning/placement for large-scale modules using b*-trees. In Proceedings of ACM/IEEE Design Automation Conference, 2003.

[77] K. S.-M. Li, C.-L. Lee, Y.-W. Chang, C.-C. Su, and J. E. Chen. Multilevel full-chip routing with testability and yield enhancement. In Proceedings of System Level Interconnect Prediction Workshop, pages 236-238, April 2005.

[78] S. Lin and N. Chang. Challenges in power-ground integrity. In Proceedings of IEEE International Conference on Computer Design, pages 651-654, 2001.

[79] S.-P. Lin and Y.-W. Chang. A novel framework for multilevel routing considering routability and performance. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 44-50, November 2002.

[80] Y.-L. Lin, Y.-C. Hsu, and F.-S. Tsai. Hybrid routing. IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, 9(2):151-157, February 1990.

[81] V. Litovski and M. Zwolinski. VLSI Circuit Simulation and Optimization. Chapman & Hall, 1997.

[82] C.-W. Liu and Y.-W. Chang. Floorplan and power/ground network co-synthesis for fast design convergence. In Proceedings of ACM International Symposium on Physical Design, 2006.

[83] M. Marek-Sadowska. Global router for gate array. In Proceedings of IEEE International Conference on Computer Design, pages 332-337, October 1984.

[84] M. Marek-Sadowska. Route planner for custom chip design. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 246-249, November 1986.

[85] R. McInerney, M. Page, K. Leeper, T. Hillie, H. Chan, and B. Basaran. Methodology for repeater insertion management in the RTL, layout, floorplan, and fullchip timing databases of the Itanium microprocessor. In Proceedings of ACM International Symposium on Physical Design, pages 99-104, 2000.

[86] K. Mikami and K. Tabuchi. A computer program for optimal routing of printed circuit connectors. In Proceedings of IFIP, pages 1475-1478, November 1968.

[87] K. W. Morton and D. F. Mayers. Numerical Solution of Partial Differential Equations. Cambridge University Press, 1994.

[88] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajatani. Rectangle-packing based module placement. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 472-479, 1995.

[89] R. Nair, C. L. Berman, P. S. Hauge, and E. J. Yoffa. Generation of performance constraints for layout. IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, 8:860-874, August 1989.

[90] B. Owens, S. Alduri, P. Birrer, R. Shreeve, S. K. Arunachalam, and K. Mayaram. Simulation and measurement of supply and substrate noise in mixed-signal ICs. IEEE Journal of Solid-State Circuits, 40, February 2005.

[91] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. In Proceedings of Combinatorica, pages 365-374, 1987.

[92] S. M. Sait and H. Youssef. VLSI Physical Design Automation: Theory and Practice. World Scientific Publishers, Singapore, 1999.

[93] P. Sarkar, V. Sundararaman, and C.-K. Koh. Routability-Driven Repeater Block Planning for Interconnect-Centric Floorplanning. Proceedings of ACM International Symposium on Physical Design, pages 186-191, 2000.

[94] M. Sarrafzadeh, D. A. Knol, and G. E. Tellez. A delay budgeting algorithm ensuring maximum flexibility in placement. IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, 16:1332-1341, November 1997.

[95] P. Saxena, N. M. P. Cocchini, and D. Kirkpatrick. Repeater scaling and its impact on cad. IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, 23(4):451 - 463, 2004.

[96] C. Sechen and A. Sangiovanni-Vincentelli. The timberwolf placement and routing package. IEEE Journal of Solid-State Circuits, 20.

[97] J. Singh and S. S. Sapatnekar. Topology optimization of structured power/ground networks. In Proceedings of ACM International Symposium on Physical Design, pages 116-123, 2004.

[98] J. Soukup. Fast maze router. In Proceedings of ACM/IEEE Design Automation Conference, pages 100-102, June 1978.

[99] A. Srinivasan, K. Chaudhary, and E. S. Kuh. RITUAL: A performance driven placement for small-cell IC's. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 48-51, 1991.

[100] W. Swartz and C. Sechen. Timing driven placement for large standard cell circuits. In Proceedings of ACM/IEEE Design Automation Conference, pages 211-215, 1995.

[101] T. Taghavi, X. Yang, and B.-K. Choi. Dragon2005: Large-scale mixed-size placement tool. In Proceedings of ACM International Symposium on Physical Design, pages 245-247, 2005.

[102] X. Tang, R. Tian, and D. F. Wong. Fast evaluation of sequence pair in block placement by longest common subsequence computation. IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, 20(12):1406-202, 2001.

[103] X. Tang and D. F. Wong. Planning Buffer Locations by Network Flows. Proceedings of ACM International Symposium on Physical Design, pages 180-185, 2000.

[104] K. Wang and M. Marek-Sadowska. On-chip power supply network optimization using multigrid-based technique. In Proceedings of ACM/IEEE Design Automation Conference, pages 113-118, 2003.

[105] S.-W. Wu and Y.-W. Chang. Efficient power/ground network analysis for power integrity-driven design methodology. In Proceedings of ACM/IEEE Design Automation Conference, pages 177-180, 2004.

[106] H. Xiang, X. Tang, and M. D. F. Wong. Bus-driven floorplanning. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, pages 66-73, 2003.

[107] X. Yang, B. K. Choi, and M. Sarrafzadeh. Timing-driven placement using design hierarchy guided constraint generation. In Proceedings of IEEE/ACM International Conference on Computer Aided Design, 2002.

[108] B. Yao, H. Chen, C.-K. Cheng, N.-C. Chou, L.-T. Liu, and P. Suaris. Unified quadratic programming approach for mixed mode placement. In Proceedings of ACM International Symposium on Physical Design, pages 193-199, 2005.

[109] M. Yildiz and P. Madden. Global objectives for standard cell placement. In Proceedings of ACM Great Lakes Symposium on VLSI, pages 68-72, 2001.

[110] M. Yildiz and P. Madden. Improved cut sequences for partitioning-based placement. In Proceedings of ACM/IEEE Design Automation Conference, pages 776-779, 2001.

[111] J.-S. Yim, S.-O. Bae, and C.-M. Kyung. A floorplan-based planning methodology for power and clock distribution in asics. In Proceedings of ACM/IEEE Design Automation Conference, pages 766-771, 1999.

[112] H. Youssef, R. Lin, and E. Shragowitz. Bounds on net delays for VLSI circuits. IEEE Transactions on Circuits and Systems, 39:815-824, November 1992.