11-8-2012

System-on-a-Chip (SoC) based Hardware Acceleration in Register Transfer Level (RTL) Design

Xinwei Niu
Florida International University, xniu001@fiu.edu

DOI: 10.25148/etd.FI13042902

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

SYSTEM-ON-A-CHIP (SOC) BASED HARDWARE ACCELERATION IN REGISTER TRANSFER LEVEL (RTL) DESIGN

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL ENGINEERING

by

Xinwei Niu

2012

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Xinwei Niu, and entitled System-on-a-Chip (SoC) based Hardware Acceleration in Register Transfer Level (RTL) Design, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Jean H. Andrian

Hai Deng

Ming Zhao

Jeffrey Fan, Major Professor

Date of Defense: November 8, 2012

The dissertation of Xinwei Niu is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Dean Lakshmi N. Reddi
University Graduate School

Florida International University, 2012

DEDICATION

I dedicate this dissertation to my dear parents and my beloved wife, Meng, for their unfailing love, support, and encouragement. Without their patience, understanding, support, and love, the completion of this endeavor would never have been possible.

ACKNOWLEDGMENTS

My deepest appreciation goes to my advisor, Dr. Jeffrey Fan, for the admirable support and guidance provided for the completion of this work. His passion and dedication are incomparable and always inspire me.

I would also like to express my deep gratitude to the other committee members, Dr. Jean H Andrian, Dr. Hai Deng, and Dr. Ming Zhao for giving their valuable time in advising and encouraging me throughout this research.

I am also thankful to the VLSI lab for providing me access to the equipment and software for my research from the very first day of my graduate study. In addition, I thank all of the VLSI lab members, especially Luis Galarza, Lilin Guo and Xiaokun Yang for their assistance. I also appreciate former VLSI lab members Dr. Wei Zhao, Dr. Priyanka Mekala, and Dr. Charles Castello for their suggestions for this dissertation.

ABSTRACT OF THE DISSERTATION

SYSTEM-ON-A-CHIP (SOC) BASED HARDWARE ACCELERATION IN REGISTER TRANSFER LEVEL (RTL) DESIGN

by

Xinwei Niu

Florida International University, 2012

Miami, Florida

Professor Jeffrey Fan, Major Professor

Today, modern System-on-a-Chip (SoC) systems have grown rapidly due to the increased processing power, while maintaining the size of the hardware circuit. The number of transistors on a chip continues to increase, but current SoC designs may not be able to exploit the potential performance, especially with energy consumption and chip area becoming two major concerns. Traditional SoC designs usually separate software and hardware. Thus, the process of improving the system performance is a complicated task for both software and hardware designers. The aim of this research is to develop hardware acceleration workflow for software applications. Thus, system performance can be improved with constraints of energy consumption and on-chip resource costs. The characteristics of software applications can be identified by using profiling tools. Hardware acceleration can have significant performance improvement for highly mathematical calculations or repeated functions. The performance of SoC systems can then be improved, if the hardware acceleration method is used to accelerate the element that incurs performance overheads. The concepts mentioned in this study can be easily applied to a variety of sophisticated software applications.

The contributions of SoC-based hardware acceleration in the hardware-software co-design platform include the following: (1) Software profiling methods are applied to H.264 Coder-Decoder (CODEC) core. The hotspot function of aimed application is identified by using critical attributes such as cycles per loop, loop rounds, etc. (2) Hardware acceleration method based on Field-Programmable Gate Array (FPGA) is used to resolve system bottlenecks and improve system performance. The identified hotspot function is then converted to a hardware accelerator and mapped onto the hardware platform. Two types of hardware acceleration methods – central bus design and co-processor design, are implemented for comparison in the proposed architecture. (3) System specifications, such as performance, energy consumption, and resource costs, are measured and analyzed. The trade-off of these three factors is compared and balanced. Different hardware accelerators are implemented and evaluated based on system requirements. 4) The system verification platform is designed based on Integrated Circuit (IC) workflow. Hardware optimization techniques are used for higher performance and less resource costs.

Experimental results show that the proposed hardware acceleration workflow for software applications is an efficient technique. The system can reach 2.8X performance improvements and save 31.84% energy consumption by applying the Bus-IP design. The Co-processor design can have 7.9X performance and save 75.85% energy consumption.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SYMBOLS AND ABBREVIATIONS

SYMBOLS	DEFINITION
ITRS	International Technology Roadmap for Semiconductors
VLSI	Very-Large-Scale Integration
CPU	Central Processing Unit
SoC	System-on-a-Chip
IP	Intellectual Property
DFT	Design-For-Test
DSP	Digital Signal Processing
ASIC	Application-Specific Integrated Circuit
FPGA	Field-Programmable Gate Array
PDA	Personal Digital Assistant
GPS	Global Positioning System
LCD	Liquid Crystal Display
CMOS	Complementary Metal-Oxide-Semiconductor
GPIO	General Purpose Input/Output
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver/Transmitter
RF	Radio Frequency
CAD	Computer Aided Design
SBP	Software Based Profiling
HBP	Hardware Based Profiling
PC	Program Counter

AMD	Advanced Micro Device
ISA	Instruction Set Architecture
HDL	Hardware Description Language
DVD	Digital Versatile Disk
MPEG	Moving Picture Experts Group
SD	Standard Definition
HD	High Definition
720p	720 progressively scanned lines
1080p	1080 progressively scanned lines
AVC	Advanced Video Coding
VCEG	Video Coding Experts Group
MB	Marco Block
ME	Motion Estimation
MC	Motion Compensation
DCT	Discrete Cosine Transform
GPU	Graphics Processing Unit
GIPS	Giga-Instructions per Second
GBPS	Giga-Bytes per Second
DCT	Discrete Cosine Transform
VCL	Video Coding Layer
NAL	Network Abstraction Layer
PAL	Phase Alternation Line
NTSC	National Television System Committee

SECAM	Sequential Color with Memory
LUT	Look Up Table
RISC	Reduced Instruction Set
LMB	Local Memory Bus
BRAM	Local Memory Block Random Access Memory
PLB	Processor Local Bus
FSL	Fast Simplex Link
XCL	Xilinx Cache Link

I INTRODUCTION

1.1 Motivation

Since the invention of modern computers from the middle of the twentieth century, semiconductor manufacturers have focused mainly on processing as much information as possible while maintaining or minimizing the execution time. In addition, as technology evolves, the size of the transistor shrinks, more transistors can be packed into a chip. Moreover, reducing the size of transistor cannot only minimize the area of the chip but also complete more tasks in a relatively short time. The devices can be more portable due to the small size of the processing unit.

According to the International Technology Roadmap for Semiconductors (ITRS) [1] and the Moore's law, the number of transistors on integrated circuits doubles approximately every two years. This trend will continue for at least another decade [2, 3, 4]. In order to benefit from the growing integrated chips, the semiconductor manufacturers continue to make great improvements on the high density of Very-Large-Scale Integration (VLSI) circuits [5]. This process not only improves the performance of modern Central Processing Units (CPU), but also other electronic devices.

Semiconductors have developed new techniques to manage the integration of many electronics components, called System-on-a-Chip (SoC). SoC refers to integrating all components of a computer or other electronic systems into a single integrated circuit (chip) [7]. The role of a SoC designer is to integrate all the needed parts into a chip to execute sophisticated functions in a relatively short period of time [8]. The whole process involves connecting Intellectual Property (IP) blocks to the system, implementing Design-For-Test (DFT) techniques, and validating the overall system level design and so on [9, 10].

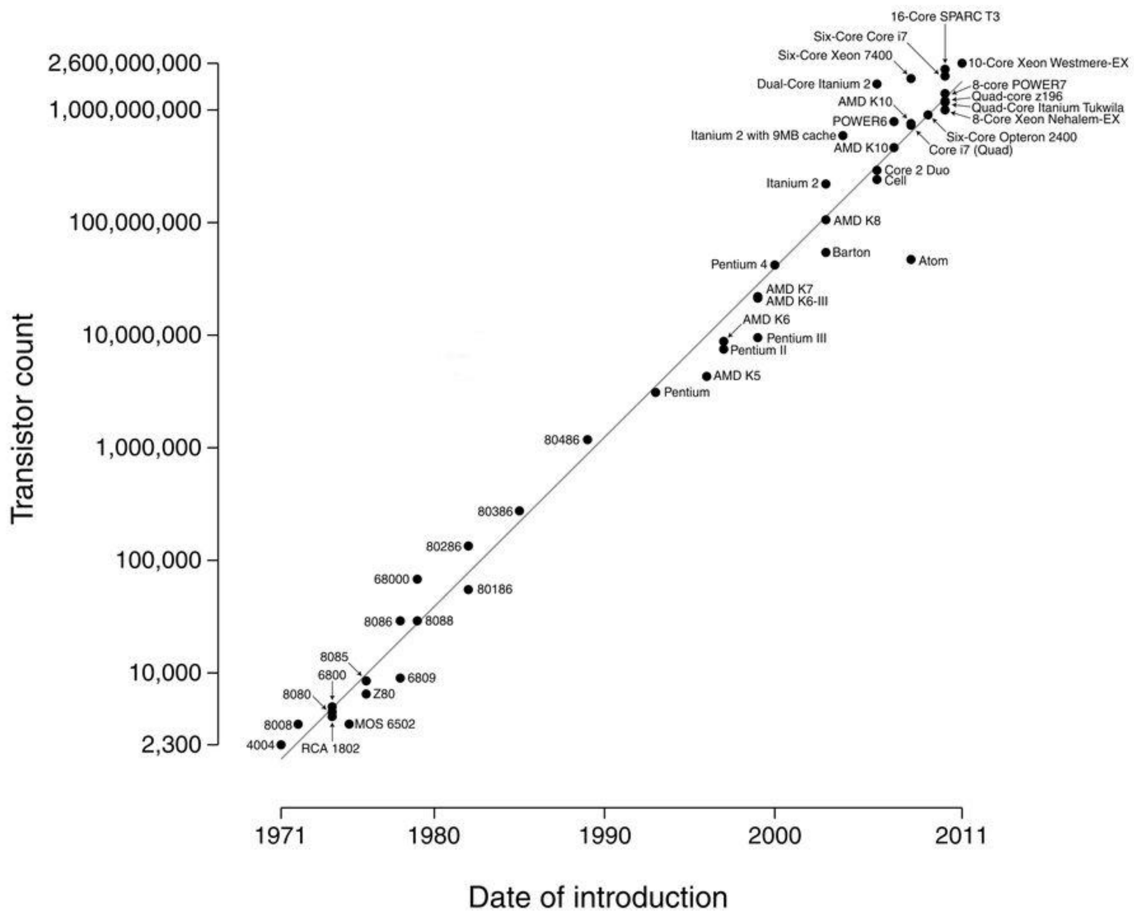


Figure I-1 CPU Transistor Counts against Dates of Introduction

However, fast execution time requires more power consumption. In the past, the focus has been the performance's gain. This has been done by increasing the frequency of a dedicated circuit. Nevertheless, semiconductors have to account for so-called "Power wall", "Memory wall", and "Instruction level parallelism wall", into design considerations.

Nowadays, performance improvement due to the increased frequency may not lead to a real efficient implementation. A designer needs to analyze the trade-off between power consumption and performance gains. An implementation that burns more power than the increased performance can't be tolerated. In other words, designers meet the "power wall." In addition, designers have to face another problem called "memory wall." They can increase the processor clock rate, but they cannot change the clock rate of a memory system to run at the same speed of the processor, hence limited by the speed of a memory system. Even though designers have built caches to improve processing speed, some of the data still have to be fetched from memory, thus limiting the real performance gain. The third problem a designer faces is "instruction level parallelism." As the clock rate is increased, the processor can execute more instructions in less time. However, the problem is that finding instructions, which can run in parallel from existing programs, is a very difficult task, especially when there are dependencies involved. Hence, increasing the clock rate does not always provide the optimal performance gain.

In this regard, what is the direction of modern system design? How to improve the system performance and keep it running at an efficient level?

1.1.1 Conventional Methods in System Design

Today's electronic systems are much more complex and have more integrated circuits. The rapid progress of silicon technology has prompted a change in the traditional processor-based design of electronic systems.

The traditional approach to the design of a system has been the independent development of hardware and software. Even though manufacturers attempt to manage the hardware and software together, they still have to implement the design from hardware and software sides separately [11].

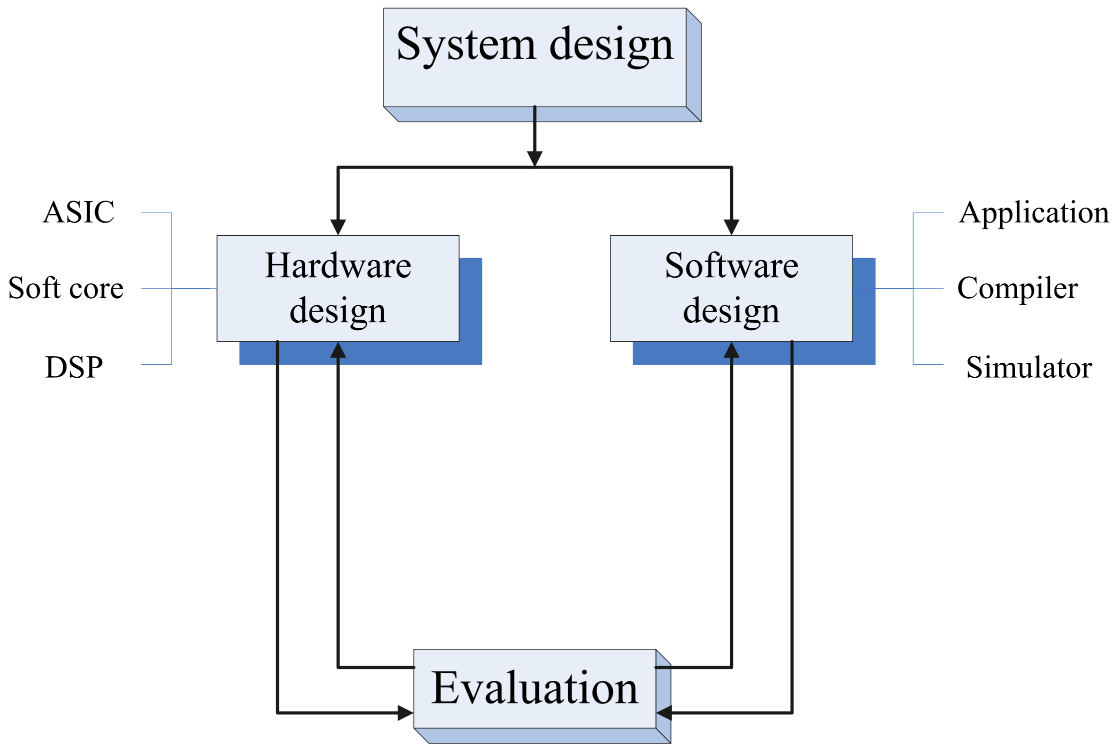


Figure I-2 Traditional System Level Design

As shown in Figure I-2, the system is analyzed and the hardware specifications are sent to hardware designers. At the same time, software designers will get the corresponding software specifications. Hardware designers can select from various implementations, one is a Digital Signal Processor (DSP) [12, 13], as a specialized processor for certain types of applications. This method is not flexible compared to other solutions. Another way is developing a specialized processing chip called Application-Specific Integrated Circuit (ASIC) [14, 15, 16] to execute specific functions to expedite the process. However, the cost of this method is very high. Alternatively, they can even use a more flexible system such as Field-Programmable Gate Array (FPGA) [11, 17] to build the hardware platforms. Software designers can develop customized compilers according to the architecture of the chosen hardware; they can also design the corresponding simulators for desired software applications. In the past, software was the only component that needed to be redesigned. However, with the current development of reconfigurable hardware, the complete system can be redesigned and evaluated.

For the traditional design process, the hardware and software work independently and test their implementations until both hardware and software are completed with their implementations at the evaluation stage. With this approach, designers have to make architectural decisions early in the process, and the high re-design cost can hardly result in an optimal design. It also requires the design team to have a wide knowledge to be successful.

1.1.2 Proposed Design Architecture

The motivation of this dissertation is to implement new design methods to improve the performance of an embedded SoC system. The proposed methods do not only allow quick and reliable system design, but also have the ability to use resources more efficiently to get an optimum solution. The workflow of the design methods consists of the following:

- Profile and analyze sophisticated software applications.
- Design the corresponding hardware accelerator and optimize it.

- Find an optimum solution by balancing the system performance, energy consumption and resource allocation.
- Design a system verification platform for the designed hardware.

System performance, energy consumption, and the cost of on-chip resources are important factors in our system design. The proposed hardware-software co-design methods include the tools used, techniques and implementations, which support the integration of hardware and software design in a more efficient way.

1.2 Research Purpose and Difficulties

The aim of this research is to improve the performance of embedded SoC systems with constraints of energy consumption and on-chip resource costs. Hardware RTL designs and optimizations are also researched. Modern SoC systems have grown rapidly due to the increased processing capabilities while maintaining or reducing the hardware circuitry. The design of an embedded SoC system is not the same as designing a general computer system. Designers have to consider a number of factors to build an efficient platform.

Based on the Moore's law [3, 4], the number of transistors on a chip will continue to increase, but current SoC designs may not be able to exploit the potential performance, especially since energy consumption and chip resource costs become two major factors. As a result, the characteristics of software applications must be identified and analyzed.

The hardware accelerator can have significant improvements on the intensive mathematical calculations or repeated functions, such as calculation of intensive algorithms and loops. The performance of a SoC system can be improved by accelerating the element that causes the calculation overhead.

An efficient system is a platform that is able to balance the trade-offs of different performance-related factors. The optimum system achieves better performance while costing less energy and resource.

1.3 Significance of the Research

The proposed methods focus on improving the performance of future embedded SoC systems. As technology evolves, embedded SoC systems have gained much more attention. Consequently, consumers are more likely to use these systems in their daily lives, such as smart phones, personal digital assistant (PDA), Global Positioning System (GPS). Meanwhile, in the next decades, manufacturers will be able to increase the number of transistors into a single die. As a result, designers must make sure the design cycles meet the market schedule. However, current hardware acceleration methods usually identify the hotspot function only by execution time or loop rounds, which are not very efficient and accurate enough. Thus, the potential performance improvement may not be fully utilized.

Nowadays, consumers are more concerned about the performance of a computer system. Therefore, the overall embedded SoC systems are the targets to be accelerated. The most significant improvement of accelerating embedded SoC systems is that they can be compared, extracted, and analyzed. This is accomplished by using the proposed hardware acceleration method to design a specialized hardware accelerator, and to improve performance while keeping energy consumption and resource costs at a minimum compared to the original implementation.

The software application is profiled and the most commonly used functions are fully analyzed. Then, a hotspot function, which is suitable to be accelerated, can be identified. After converting this hotspot function into a hardware accelerator, the system specifications, such as speed, energy, and resource consumption, can be generated. The hardware optimization method is used to optimize the designed hardware accelerator running on the SoC system. The trade-off of system performance, energy consumption, and resource costs among different hardware accelerators are balanced and the optimum solution is chosen for the SoC system.

1.4 Structure of the Dissertation

This dissertation is structured as follows: Chapter 2 introduces the background information for the proposed methods of the dissertation. An overview of the SoC system, H.264 video Coder-Decoder (CODEC) standard, software profiling theory, and hardware acceleration techniques, are also introduced. Chapter 3 explains the major design details of the proposed method for embedded SoC system architecture, from understanding the characteristics of software applications and finding the critical path for a design, to spotting the desired function to be implemented in hardware acceleration. A detailed overview of the hardware-software co-design workflow used in this dissertation is introduced. Chapter 3 also explains hardware platform design for the aimed SoC system structure. The method of implementing a software function to the hardware accelerator is explained. Hardware optimization methods are used in the design to gain better performance and reduce resource costs. Different hardware designs are used, compared, and analyzed. Chapter 4 describes the software experimental platform. H.264 video CODEC standard is used as a target application to be accelerated. A detailed workflow of software methods is validated by preparing, profiling, and analyzing functions of the software. The hotspot function that incurs system overhead is identified. Chapter 5 describes the hardware platform which is used in this research. After analyzing the software application and finding the desired hotspot function, hardware accelerator is implemented and wrapped as an IP. Hardware IP is mapped to the SoC system by guiding the data flow through hardware instead of the pure software method. Thus, the system can have performance improvement and reduced energy consumption. The hardware-software co-design method is demonstrated based on the platform provided. Different hardware accelerators, which cost different amounts of resources, are also evaluated for the optimum solution. Chapter 6 expands the proposed contribution to the Integrated Circuit (IC) design workflow. The Sobel operator, which is widely used in remote control system, is verified as an example on the created verification platform. The RTL design of the edge detection part is optimized to improve the system performance and reduce resource costs. Finally, Chapter 7 concludes the dissertation and summarizes further plans and suggestions for this research.

II BACKGROUND

2.1 System-on-a-Chip (SoC) Design

2.1.1 Evolution of System-on-a-Chip

The System-on-a-Chip (SoC) is a system with several pre-designed and pre-verified blocks on a single chip. These blocks are often called Intellectual Properties (IPs). IP cores could be embedded processors, memory blocks, interface blocks, analog blocks, digital blocks, or some other blocks that deal with specific purpose functions. Software components will also be provided, such as operating systems, software libraries and device drivers [18, 19], etc. The first true SoC system was a Microma watch in 1974, when Peter Stoll integrated Liquid Crystal Display (LCD) driver transistors and the timing functions onto a single Intel 5810 Complementary Metal-Oxide-Semiconductor (CMOS) chip [20]. From then on, as the number of transistors continues to increase on integrated circuits, the industry continues to make great improvements in the density of Very Large-Scale Integration (VLSI) circuits.

Figure II-1 is a general SoC architecture. Modern SoC systems usually include a bunch of functional modules, such as a Central Processing Unit (CPU) as the key block, memory controller, and the physical memory as storage blocks. Huge data could be stored in the memory, so that the CPU can fetch and put them back, if necessary. The SoC system usually has a system bus, which is used for communications among different blocks. Currently, there are several types of buses and each bus type has its own bus protocol. Some SoC systems have co-processor blocks and Digital Signal Processing

(DSP) blocks, which can relieve the processing burden of the CPU. The SoC system also has other peripherals like General Purpose Input/Output (GPIO), Ethernet, Universal Serial Bus (USB), etc.

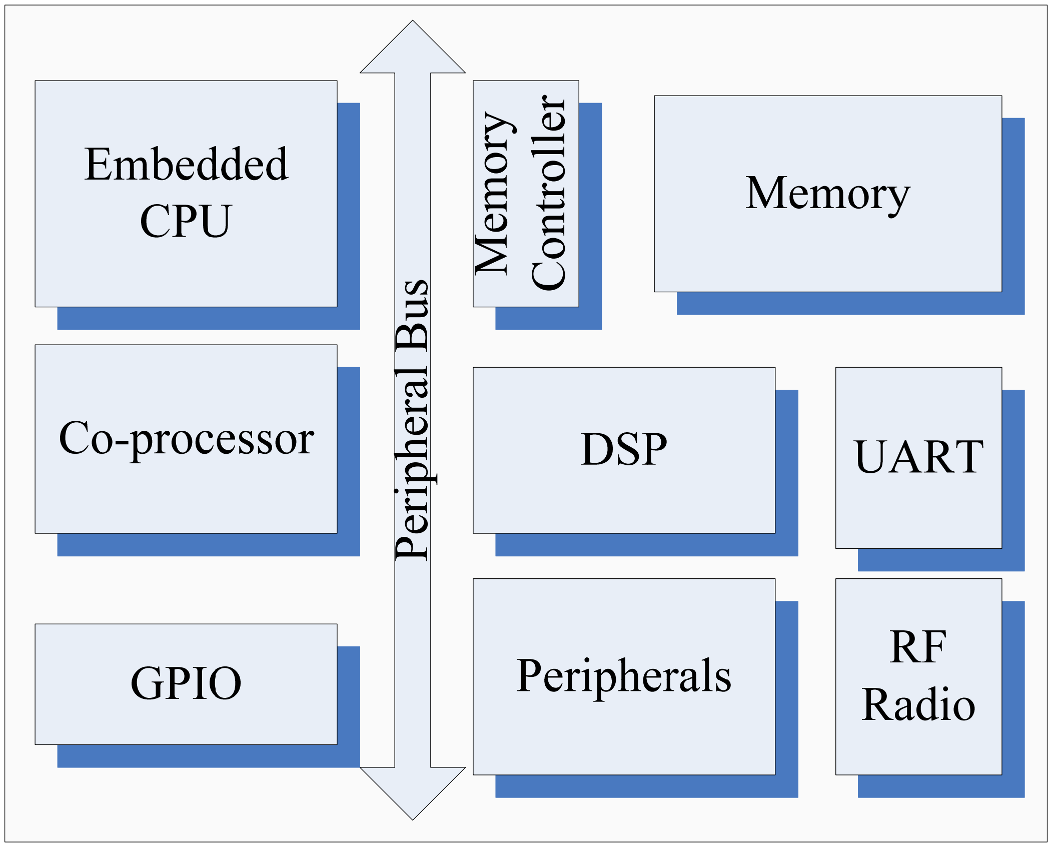


Figure II-1 Gernal SoC Architecture [10]

2.1.2 Challenges for SoC Design

In early years, the microprocessor has only around two thousand transistors and can only perform limited logic functionalities. However, a modern SoC system is composed of billions of transistors, which could perform a wide range of functions [8]. Manufacturers always try their best to fit the Moore's law, so that the integrated transistors on a chip are increasing constantly. However, people cannot just simply squeeze more and more components onto a single chip. Now designers face a gap between what they can theoretically design and what they can practically implement. Figure II-2 is the productivity gap, which explains the annual growth rate of logic transistors on a chip is 58%, while the productivity growth rate in terms of logic transistors per month is only 21%. The gap between these two lines will continue to increase in the future [21].

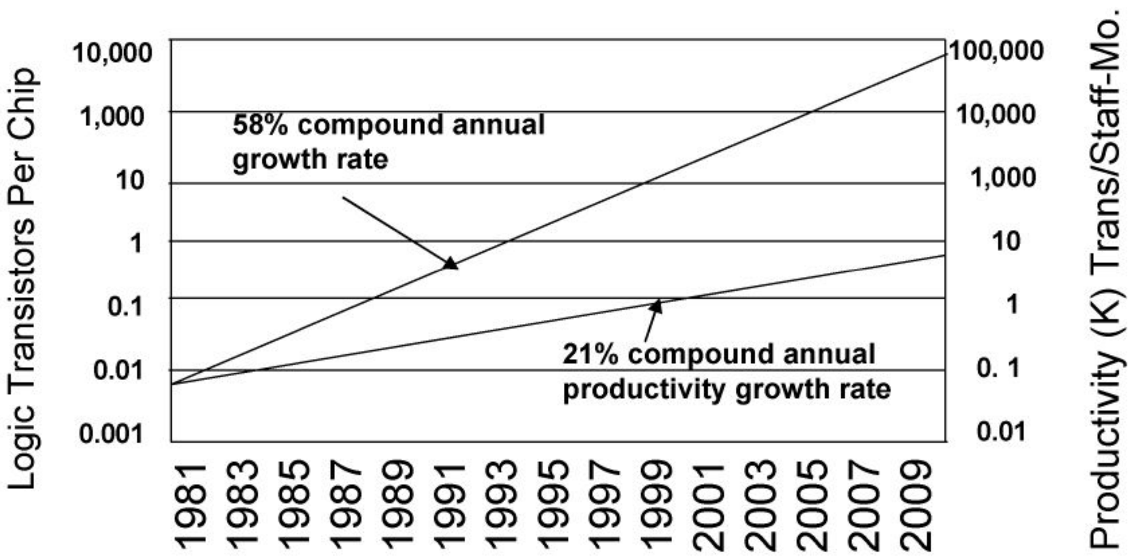


Figure II-2 Productivity Gap [ITRS]

As the technology evolves, the SoC has pushed the development of industry-wide standards. Some examples are bus standards [22], bus interface standards [23], IP formats, protection [24], and test wrappers [25]. At the same time, we still face several challenges [21]:

- Design productivity. The designs are getting more and more complex, thus causes the poor design productivity and the high design cost. Transistor counts and speed are increasing exponentially. Moreover, there are demands for increasing design functionalities and keeping the resource costs at a low level. Nowadays, more and more blocks are being embedded on the SoC system, so the influence among different components is increasing greatly.

- Power consumption. The power has become a critical issue in nanometer technologies. The aspects like noise and leakage current that are previously ignored are becoming the critical issues for a design. Even though designers can easily increase the transistor counts and the chip speed, the dynamic and static power consumption will also increase because of the greater leakage current. The research also shows that the static power could be higher than the dynamic power [26].

- Manufacturability. Manufactures can integrate more transistors onto a single chip. However, this also generates problems in the chip yield, reliability, and testability. Currently, different design processes are making the design unpredictable, and hence require more Design-For-Test (DFT) methodologies. The new DFT should have the abilities of improving test coverage, reducing test time, etc., which will lower the overall cost of the SoC design.

2.1.3 Future System-on-a-Chip Design Trends

For the future SoC system design, the following areas are being discussed:

- Power efficiency. The power consumption will not decrease when the gate density continues to grow. Strategies, which can lower the power consumption, have been discussed. People can use some methods to manage the idle cycles of a system to save the power [27, 28]. Another way is to reduce the power consumption of the clock distribution. For example, designers can use only one clock frequency, but different blocks may use different phases of this clock [29, 30, 31]. Furthermore, temperature-aware design is also a choice. Temperature variations may lead to electronic failure, timing uncertainty, and even degrade system performance, so it is important to integrate different "hot" and "cool" blocks together for a power-efficient design [32]. More power efficient techniques, which can manage the system temperature, are discussed in [33, 34].

- Multi-processor SoCs. The multi-processor SoC system is a solution that will not only increase the processing capabilities, but also keep the power consumption at an acceptable level. It has become more and more popular in the modern system design [35, 36, 37, 38]. Processors embedded on the single chip do not need to be the same type. An efficient way is to arrange specific processors perform some sophisticated and frequently-occurring functions [39, 40, 41, 42].

- Reconfigurable Blocks. Currently, some SoC platforms start using reconfigurable logic operations and interconnected blocks. This will provide not only the flexibility, but also the high performance [43, 44].

- Design for Test and Verification. When manufacturing integrates millions of transistors onto a single chip, designers will take much more time to test and verify it as a whole unit. When there is even a small change in one component, designers need to re-test and re-verify the entire chip. One way to avoid this is to create clear boundaries and channels among different blocks, so that changing one block will not affect others. Another way is to build self-test function in each block to shorten the verification time [45, 46, 47].

2.2 Software Profiling for Software Partition

2.2.1 Why Profiling?

Over the past decade, the embedded SoC system has grown rapidly due to the increased processing ability and transistor density on the circuit. At the same time, the area of the chip is still small enough. Users make use of these kinds of devices for communication and entertainment activities. Some examples are cell phones, digital cameras, portable game consoles, and so on. Even in large industrial companies, SoC systems are usually used as controlling devices, or some medical instrumentations.

Moore's law guides the industry run at a high speed, so it proposes great challenges to designers, who pay more efforts during the design process. Customer demands for advanced SoC devices are increasing, and this will result in shorter design cycles and time-to-market schedules. In order to keep the chip area in a relatively small size, and at the same time control the power consumption at a low level, designers must use more powerful Computer-Aided-Design (CAD) tools to meet more and more complex system requirements. There is a demand that the final product should have the optimized partition of hardware and software. This is why profiling plays such an important role because it determines which component is the bottleneck of the whole system. The profiling process help designers to create a proper balance between hardware and software in a system. Thus, a better overall system performance with little overhead can be achieved [49, 50, 51].

2.2.2 Classification of Embedded Software Profiling Techniques

Performance analysis platforms for software have been researched for years. As shown in Figure II-3, there are three types of profiling methods: Software Based Profiling method (SBP), Hardware Based Profiling method (HBP), and FPGA based profiling method. They have different profiling abilities and support applications written in different languages [52, 53].

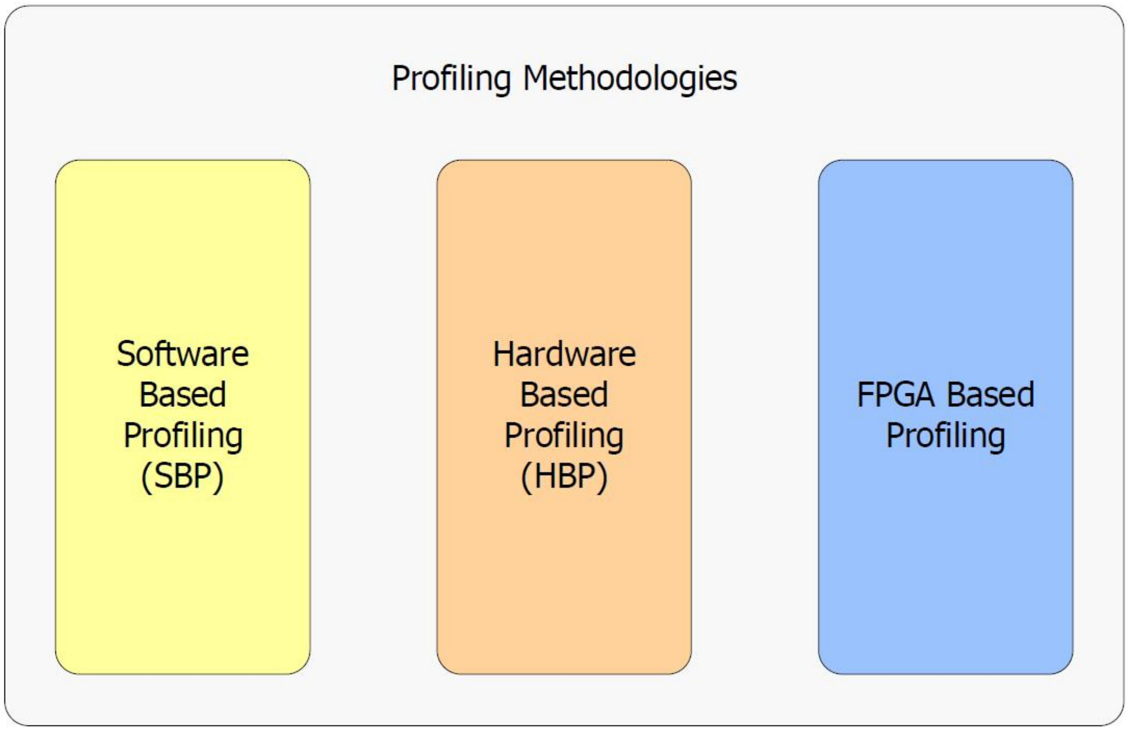

Profiling Methodologies
Software Based Profiling (SBP)
Hardware Based Profiling (HBP)
FPGA Based Profiling

Figure II-3 Profiling Methods Category

Profiling tools are used to measure the performance of a system. They can monitor the time needed to finish particular functions, which run in the target processor of the embedded SoC system. Normally, profiling tools are used to detect the bottleneck of a software application. This will help embedded SoC system designers to evaluate and optimize the design in order to meet system requirements.

- Software based profiling (SBP) tools. SBP tools are mostly used for evaluating the performance of applications written in the programming language. The virtual simulation is one way to profile software applications. It simulates the application code running in the virtual environment. Designers benefit by viewing the entire data flow movement during the simulation. The virtual simulation method keeps tracking all of the processes inside the microprocessor. Therefore, designers can easily identify the bottleneck of the application. However, the virtual simulation method will be very slow to simulate large software programs. It will lead to inaccurate profiling data on the performance of each function [54]. On the other hand, the instrumentation code insertion [55] is a widely used profiling method on Linux and UNIX workstations [56, 57]. It inserts instrumentation code to fetch the data of the running CPU, and reads the number of program counters (PCs) to collect the exact number of the called function. The execution time of a program can also be gathered by reading the value of PCs at a specified time frame. However, because the execution time greatly depends on the frequency of the PC, the accuracy of the profiled execution time is not guaranteed [58].

- Hardware based profiling (HBP) tools. Advanced processors are processors, such as Sun Ultra SPARC, Intel Pentium Processors, and Advanced Micro Device (AMD) Processors. They utilize on-chip hardware counters to profile the performance of the CPU [59, 60, 61]. These kinds of hardware counters aim for some specific events when the applications are executed, such as cache hit, cache miss, memory accesses, etc. It is a better choice to use hardware counters because they don't touch the application code. The other benefit is this kind of counters add little performance overhead because the data is collected during the run-time. Nevertheless, it has several drawbacks: First, hardware counters may need to be re-configured sometimes in order to profile another type of event. Second, using the HBP method will finally use the strategy of the SBP, which may cause the same problems. Third, the number of hardware counters is limited, so that it will be a time consuming work when many events need to be monitored. Fourth, processing the hardware counters will activate the system interrupt services, the number of services called will be added to the quantities of events, so the data collected may not be accurate enough.

- FPGA based profiling tools. Due to the rapid evolution of modern technologies, the FPGA provides more and more resources to be used. FPGA based embedded SoC systems usually consist of a soft-core processor, buses, and several kinds of Intellectual Properties (IPs). Different manufacturers deliver their own soft-core processors. These processors are used as the central component of embedded systems [63]. Currently, two major FPGA vendors are Altera [64] and Xilinx [65], with their soft-cores named Nios II [66] and MicroBlaze [67]. The FPGA based profiling method also uses the soft-core to collect the performance data. On-chip profiling hardwares gather all the needed data, when the application code executes on the soft-core processor. FPGA based profiling tools can keep the latency and the performance overhead at a minimum level. It requires little processor computations because it usually doesn't use the sampling technique.

2.3 Hardware Acceleration

People are always trying to run applications as fast as possible. However, modern processors can not always gain the performance improvement by just increasing the frequency of the CPU. Even with the multi-processor architecture, some sophisticated applications, which require huge data calculation, still can not meet the design expectations. Usually, software applications written in C language are sequentially processed. In this case, if the application is calculation intensive, it will cost more time and consume more power to complete tasks.

Therefore, people begin to implement the special purpose engine called hardware accelerator to execute a dedicated function of the application. Notable performance improvements can be achieved by using hardware accelerators in the computer aided design. This technology also improves the computer aided design methodology in the past years [68, 69].

The hardware acceleration also has trade-off. Implementing the whole algorithm to hardware costs the least time, but it is also the least flexible and the most expensive way. Although a general purpose computer provides great flexibility and the least cost, it is the slowest method for software applications. Thus, it is important to balance the trade-off of a system, when constructing a hardware accelerator. The solution will be different depending on different problems, the design environment, methodologies, etc. [70,71].

Currently, there are three major types of hardware acceleration methods:

- Implementing multiple instructions at the same time to increase the performance. Several instructions are compiled into one instruction to be the processor's Instruction Set Architecture (ISA). This is shown in Figure II-4.

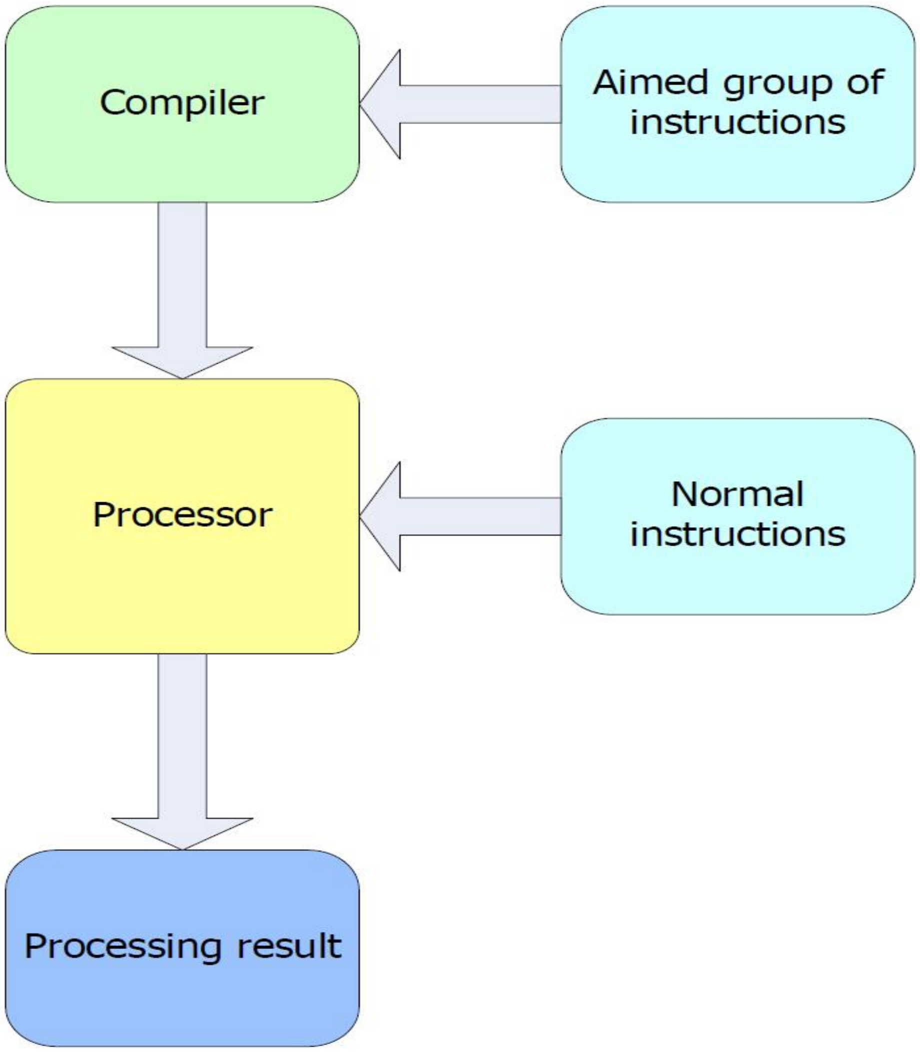


Figure II-4 Hardware Acceleration using New Created Instruction

For the hardware acceleration method using new created instruction, designers need to choose some aimed instructions. These aimed instructions can represent some specific functions, and the functions are chosen because they have high percentage occupations, when software applications are executed.

The aimed instructions are sent to the compiler for the corresponding ISA. Multiple instructions are compiled into one or few instructions with the same functionalities. After that, compiled instructions are sent to processor combined with the normal instructions. The execution time of a CPU to process one instruction is fixed, so the fewer the instructions, the less of the execution time. Meanwhile, the energy consumption can also be reduced.

- Implementing the hardware acceleration part as a co-processor. The designed hardware accelerator is directly connected to the processor through processor's high speed interfaces. The designed co-processor communicates with the CPU directly. This is shown in Figure II-5.

For the hardware acceleration method using the co-processor design, designers also first identify highly used functions of the whole software application. These functions are often called "hotspots". After successfully profiling the application, designers transform the hotspot function to the corresponding hardware using hardware description languages, such as Verilog HDL [72] or VHDL [73]. The designed hardware part is wrapped as an IP and attached to the processor directly through processor's high speed interfaces.

- Connecting the designed hardware accelerator to the processor via a system bus. The hardware accelerator communicates with the processor through the system bus. This is shown in Figure II-6

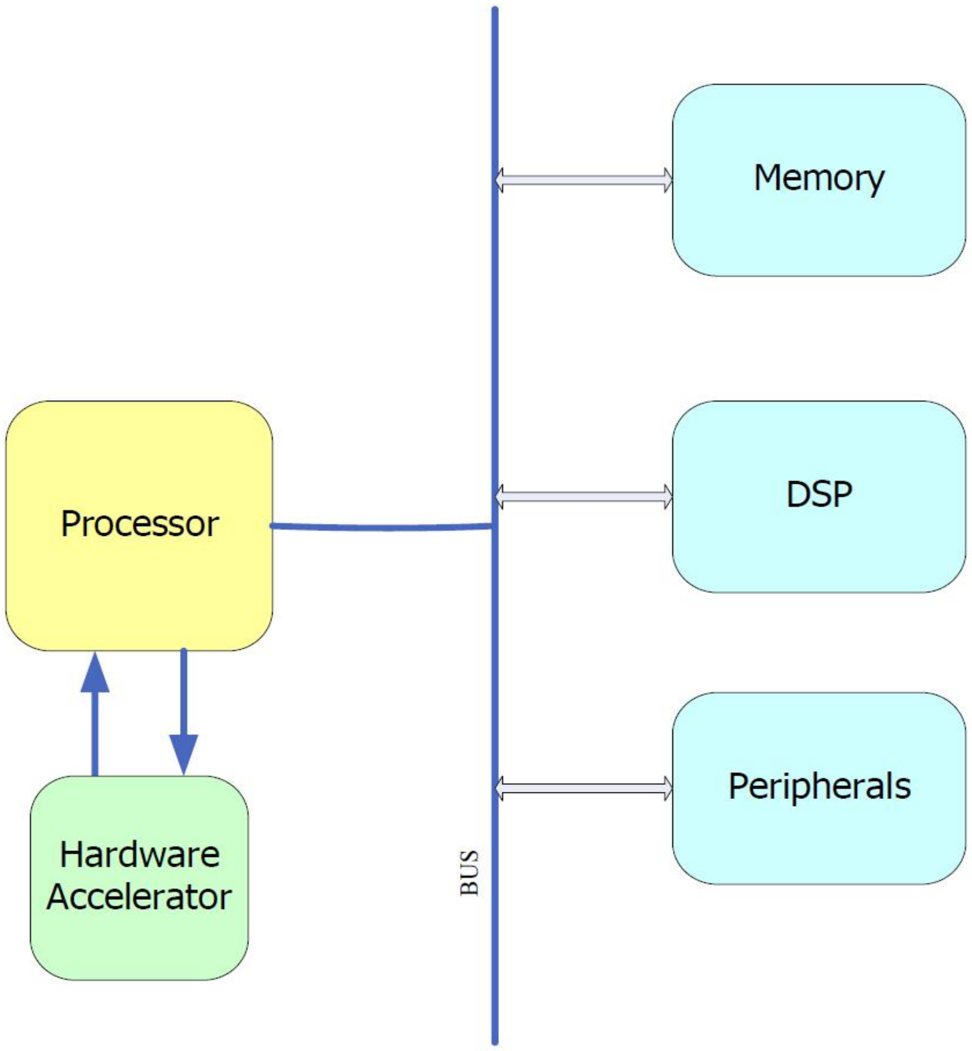


Figure II-5 Hardware Acceleration using Co-processor Design

For the hardware acceleration method that connects IPs to the system bus, designers do the same things as the co-processor design method for the first several steps. Designers profile the software application, and then use hardware description languages to convert a hotspot function to a specific IP. The numbers of IPs that can be generated are not limited. After that, designers will not directly connect the IP to the processor but to the system bus.

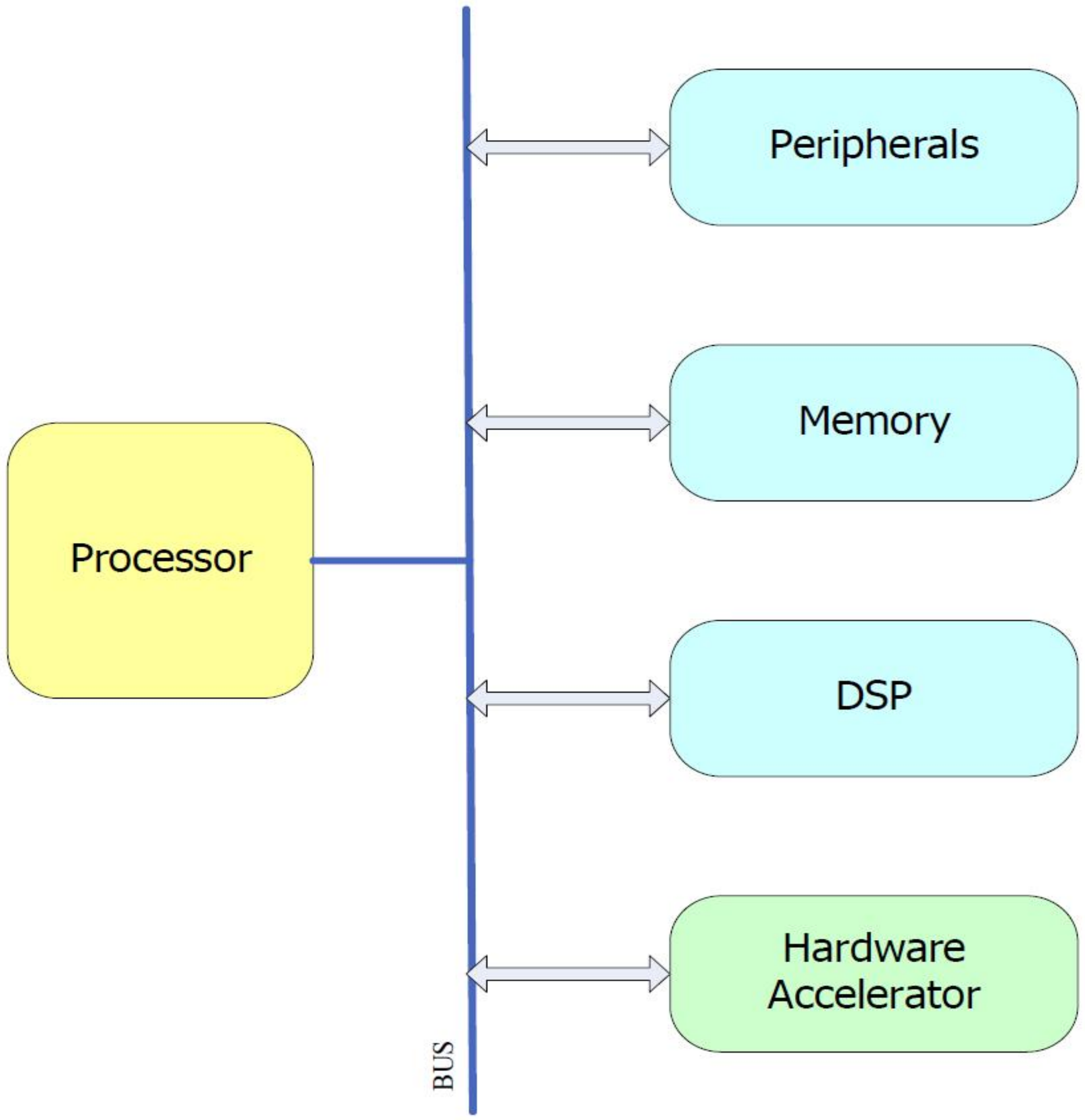


Figure II-6 Hardware Acceleration using System Bus

There are several benefits to connect IPs in this way: First, this method resolves the communication issue between the IP and the off-chip memory. If processors send instructions to the IP for execution and the processed data is stored in the off-chip memory. Then, the IP can directly fetch data from the memory. This is very efficient especially when large data sets will be processed. Many interactions happen among the memory, the IP, and buses. In this case, the CPU can have spare time to acknowledge other requests from other peripherals or devices.

In conclusion, hardware acceleration methods are evolving at a rapid speed. Every method has its own advantages and disadvantages. In order to identify which method should be used, designers must consider the system configuration, the goal of the project, the complexity of the design, etc. The correct decision should be made based on the full evaluation of demands.

2.4 H.264 Video Coding Standard

2.4.1 Video Coding Standard H.264

Today's video technologies have progressed rapidly in the last fifty years. People are enjoying the high speed internet throughout the world. The storage capacities of current electrical devices are growing. This evolution will not stop. People want to watch movies at home just through their internet connection; they share video streaming with their family; and they want to store all of their movie records on a single disk, etc. However, nowadays a single Digital Versatile Disk (DVD) can only store a few seconds of the raw video at the television-quality resolution and frame rate [74]. This is why there is a great need for fast efficient coding and lower energy consumption for video compression constructs.

In 1990s, the Moving Picture Experts Group (MPEG) developed the MPEG-2 video compression coding standard [75, 76, 77]. This famous video standard was very popular in the past 20 years. Most of the digital televisions (TVs), TV broadcasting, and movie DVDs are made by this video standard.

The ever evolving technology has led us to a new digital era. The Standard Definition (SD) TV is being replaced by the new High Definition (HD) TV [78]. First of all, the 1080 progressively scanned lines (1080p) used for an HD video contain more than 6X of the pixels found in a 480p SD video. In addition, today's computer system is much more complicated and powerful than those in 1990s. Obviously, the old MPEG-2 standard is not an efficient video standard any more for current novel technologies. This is why we have a new video coding standard. It is called H.264/MPEG-4 Part 10 advanced video coding (AVC) or just H.264 [79].

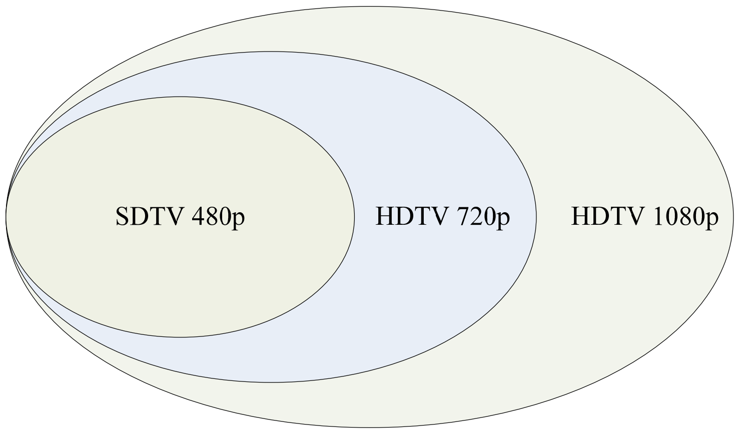


Figure II-7 Frame Sizes of SDTV and HDTV

The H.264 is defined by the Video Coding Experts Group (VCEG) and the Moving Picture Experts Group (MPEG). H.264 standard has been proven to be at least two times more efficient than its predecessors, MPEG-2 and H.263 [80, 81]. In this way, H.264 only costs half of the bandwidth compared to MPEG-2, when both of them transfer the same video, especially for the high definition video source like HDTV.

The scope of video coding standardization is shown in Figure II-8. The original video will go through an optional pre-processing step in case the sender needs a personalized video. Then, the video will be encoded and transmitted over the communication network. After arriving at the destination, the video is decoded and the user can do an optional post-processing step to convert the video to the desired format, or perform some other technical processes.

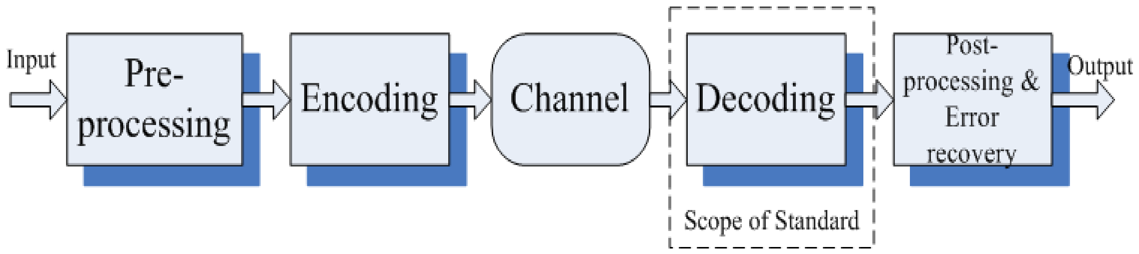


Figure II-8 Scope of Video Coding Standardization

For the H.264/AVC standardization, only the central decoder is standardized by defining the syntax and semantics of the bit stream as well as the processing that the decoder must conform. Thus, every decoder produces similar output data when given the same video. This allows manufacturers maximum freedom to optimize their products in pre-processing, encoding, and post-processing [82, 83].

2.4.2 H.264 CODEC Algorithm

The original video data often include a lot of redundancy. The redundancy only has limited influence if it is removed. People develop video compression algorithms to eliminate video redundancy. Under this way, the video source will be compressed to small size while the quality is kept at an acceptable level. Video signals have two significant types of redundancy: time-domain redundancy and spatial domain redundancy. Some video compression algorithms also apply the arithmetic redundancy elimination to the coded data.

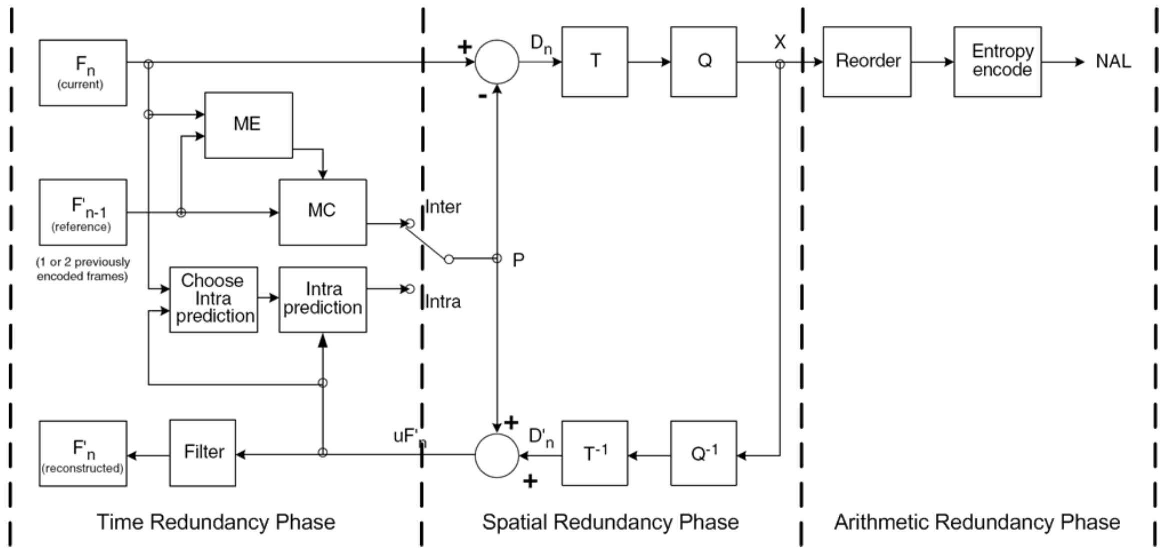


Figure II-9 H.264 CODEC Algorithm in Phases

Figure II-9 shows the H.264 coder/decoder [74]. Generally, H.264 standard compresses media in Macro Blocks (MBs) rather than in frames. One MB is a 16×16 pixel block. When the process starts, the H.264 encoder will separate current video frame into numerous 16×16 MBs, and the MBs will be processed one by one.

There is a very important module called Motion Estimation (ME) in Time Redundancy Removing Phase [84, 85]. After splitting frames, MBs from current and several previous frames will be sent to the ME module. A lot of computation and self-analysis processes will occur in this module, then the closest MB within the previous frames which could be used to represent current MB will be found by ME module. The relative position-Motion Vector (MV) and absolute difference called Motion Residue (MR) will also be generated. Then all information is sent to the next module, which is Motion Compensation (MC) [86, 87].

In Spatial Redundancy Removing Phase, the Discrete Cosine Transform (DCT) is used as a key component to differentiate the high/low frequency parameters of the residue of an MB. The energy from the low frequency domain will occupy most of the energy of these frequency parameters. In order to eliminate the remaining high frequency part but still keep the low frequency part, a quantization map will be applied to the DCT results. Then, the arithmetic coding (e.g., Huffman coding) is used to eliminate the arithmetic redundancy [88].

2.5 Field-Programmable Gate Array (FPGA)

The rapid evolution of technology gives engineers more challenges to design the low cost, high performance devices in a shorter design cycle. Currently, the FPGA is viewed as a potential alternative to provide a quick hardware instantiation of computationally intensive algorithms. FPGA platforms provide a better solution for computation intensive applications due to their powerful abilities to achieve quick and reconfigurable designs [89]. FPGA designs usually involve the following steps shown in Figure II-10.

- Design Entry. Implement the logic circuit in the Hardware Description Language (HDL) or a schematic design editor.

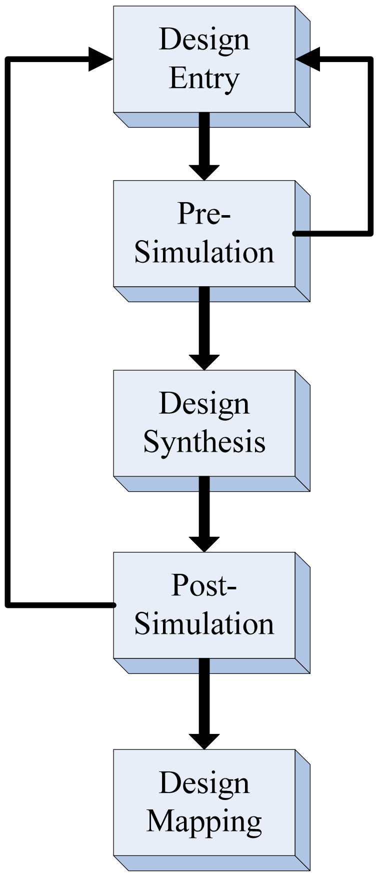


Figure II-10 Implementation of Logic Design with FPGA

- Pre-Simulation. Also called the behavioral simulation, this is used to verify the HDL description against the corresponding behavioral model.

- Design Synthesis. Usually logic synthesizers are provided by different manufacturers. The synthesis process transforms designs into machine-readable netlists. Netlists contain descriptions of various logic gates and interconnections.

- Post-Simulation. The post-simulation simulates the design with switching time and propagation delays taken into account. For large designs, this step is very important.

- Design Mapping. The mapping implements the design for the specific FPGA device. Bitstreams will be generated from netlists. Then, the generated bitstream can be downloaded to the target board for further verifications.

In pre-simulation step, designers verify whether the function of the design is correct. If not, designers must re-check the design and make some modifications. Even though the design passes the pre-simulation step, it may still fail in the post-simulation step. This is because in the post-simulation, the gate delay and switching time will be added to emulate the real circuit. If the design fails in the post-simulation step, designers must go back to the design entry to verify the design scheme.

2.6 Summary

Chapter 2 focuses on the background review of the modern embedded SoC design. SoC systems become more and more powerful as our industry technologies evolve. Many powerful components are integrated in the modern SoC system. Challenges of modern SoC system design are design productivity, power consumption, manufacturability, etc. Then, trends of the future SoC designs are introduced. Power efficiency is an important factor for SoC designs, especially the embedded SoC design. Thermal control will be a hot issue in the future design. In order to make a powerful SoC system, multi-processor SoC systems will be alternatives. The usage of the reconfigurable block and the new design for test process will be validated in the future. A review of current profiling methods and various profiling tools are introduced in this chapter. Basically, there are three types of profiling methods, such as SBP, HBP, and FPGA based profiling.

Hardware acceleration methods are also covered. Three major hardware acceleration methods are introduced, including the modification of the processor's ISA, the system Bus-IP design, and the co-processor design. As the most popular video CODEC, H.264 plays an important role in our daily lives. This chapter introduces the H.264 standard and the applied algorithm. The workflow of FPGA designs are also studied in this chapter.

III SYSTEM DESIGN METHOD

3.1 Methods of System Design

3.1.1 SoC Top-down Design

As the evolution of the FPGA platform continues, it has more power and capabilities to be constructed as a SoC system. A typical SoC system usually contains traditional microprocessors, which could be hard processor or soft processor; on-chip system buses, which connect to different components on the chip; a variety of I/Os, including different types of interfaces; some SoC systems even have application specific components, which could accelerate the processing speed. All these provide an opportunity for SoC designers to deal with a wide range of applications.

Table III-1 Hardware Solution vs. Software Solution

	Advantages	Disadvantages
Hardware solution	Faster solution if running parallel computations.	Applications specific; hard to modify, or reuse.
Software solution	Flexible; easy to be re-programmed.	Relatively slow; depends on hardware configuration.

The SoC system design, especially the embedded design, is different from pure hardware design. For embedded SoC design, it is a kind of hardware-software co-design idea. Generally, the software runs in the on-chip microprocessor on the hardware platform. Hardware offers fundamental computation abilities. Meanwhile, software uses these abilities to explore more application possibilities. Table III-1 shows the comparison of hardware and software solutions. For hardware solution, it is the fastest way for running parallel computation functions. However, the hardware solution is an application specific way, which means it is difficult to be modified if it needs more optimizations. On the contrary, a software solution is very flexible if designer needs to modify it. But software solution is usually relatively slow, and the speed of software solution is also limited by hardware configurations.

3.1.2 Embedded SoC Design Challenges

As we discussed in Chapter 1, a SoC system is composed of different components, which play certain functions in the system. When our industry peers push the technology evolution to a new level, bottlenecks of SoC system are found. For example: processor speed, memory bandwidth, power consumption, etc.

Currently, people usually implement specific computational component to resolve the problem. For example, Digital Signal Processing (DSP) chips are used for signal processing purpose. Graphics Processing Units (GPU) can be used to process the high volume of graphic data. Application-Specific Integrated Circuit (ASIC) is also a choice to make a specific purpose chip in order to execute some special purpose functions. More recently, when we can build more transistors on a single chip, FPGA has been used to be the host of modern SoC systems.

The FPGA is the most flexible platform when it is constructed as a source system. The system could include hard processor or soft processor, on-chip buses, off-chip memory, and a variety of I/O interfaces. People could use an FPGA to improve the system performance.

However, the FPGA is used as a hardware platform historically to design logic circuits on a board. If it is used as a SoC platform, then the software code running on the on-chip embedded processor will force the design method to effectively use all of the components on FPGA boards. This will propose challenges for the current design flow.

One concern is that the design complexity has increased. Former hardware design flow hasn't been changed for a long time. People use Hardware Description Languages (HDLs) to map the logic cells on FPGA boards to execute dedicated functions. When designing hardware using HDLs, which is a low level language, people must pay attention to not only function level of design, but also complete details of the design architecture including timing constraints. The more components to be integrated onto an FPGA board, the more complexity it creates.

Another concern is the acceptance of proposing a new design method. From the designer's perspective, the new design shouldn't be totally new to them, because that will cost a lot of time to learn and get familiar with it. A better design method should use designer's familiar routine, make them feel comfortable and same even though the implementation methods are different.

3.1.3 Hardware-software Co-design

With the emergence of FPGA platforms, which could contain hard or soft processors, people can use a complete end-to-end tool set to implement system designs. As mentioned before, traditional SoC designs often require engineers from both hardware and software areas. When the system is defined, the hardware platform will be provided. After that, software engineers can use this hardware platform and the tool suites to develop the software code. Then, software and hardware will be evaluated at the last. As an overall SoC system design method, it will produce the high possibility of errors, when software and hardware engineers communicate with each other. Besides, the design team must have a board skill to undertake the job.

In the traditional method, people can only modify the software code if the design cannot meet technical specifications. This has motivated us to develop a new design flow, which allows quick and reliable system designs. It allows designers to modify the software and hardware efficiently to get the optimal solution. These hardware-software co-design methods, including techniques, tools, etc., can fully support the hardware and software integration during SoC system designs.

Figure III-1 is the workflow of hardware-software co-design. In the new method, we use a hardware-software co-simulation at the second stage. The design will be partitioned from high level, the software part and hardware part will be simulated together, if results couldn’t meet the system requirement, designers can modify the design at an early stage.

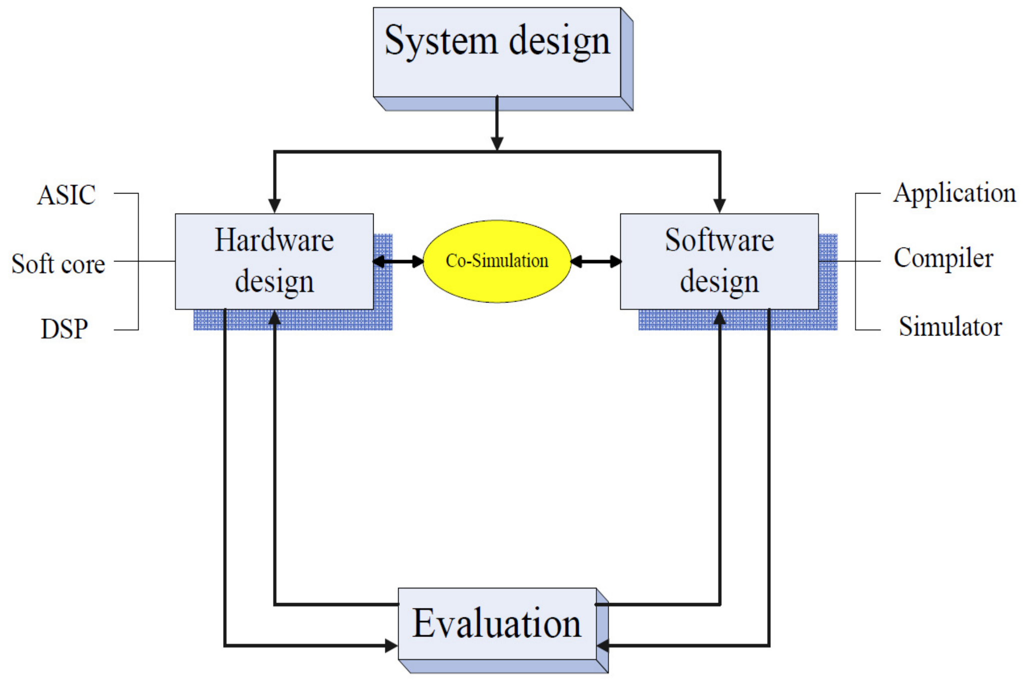


Figure III-1 Hardware-software Co-design

Function, architecture, and performance are three key aspects of SoC designs (as shown in Figure III-2). When the function and architecture of one application are made, the system can definitely achieve some particular performance; when the application function and performance are made, the designer can make the corresponding architecture to meet the requirement. So, with the new method, that hardware and software could be co-simulated at an early stage, designers can get a complete understanding of the system and make modifications and optimizations before the final chips are made. Under this way, the design complexity drops and the design cycle decreased, which can push the product to the market earlier than before. In the meantime, the research expense will also drop a lot. It can save millions of dollars before the design is made into a chip.

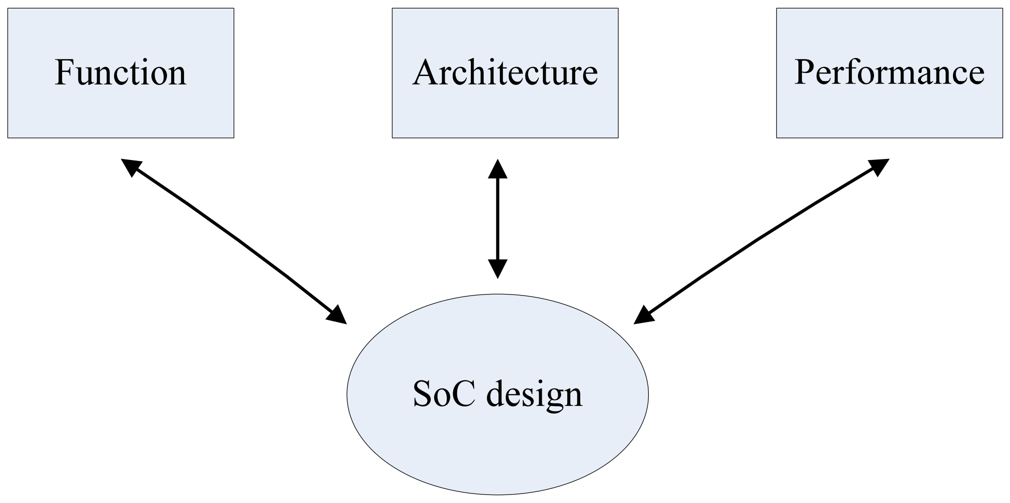


Figure III-2 Function, Architecture, and Performance for SoC Design

3.1.4 Framework of Research

Before proposing our method for hardware acceleration design, we must answer the following questions:

- Why do sophisticated applications running on embedded SoC systems have performance overhead and how to measure it?

  Performance bottlenecks may be caused by functions of sophisticated applications, which are invoked frequently. The performance overhead of applications running on an embedded SoC system could be profiled and analyzed by applying analysis tools to obtain software behaviors.

- How to accelerate an embedded SoC system after performance overhead has been identified?

  The corresponding element of software applications could be isolated and analyzed. The hotspot function could then be identified. Depending on the characteristics of the function, the hardware acceleration platform may perhaps be utilized to accelerate this kind of functions. Thus, the overall performance overhead should be reduced.

Then we can summarize the framework of this research in Figure III-3 [123], which provides a schematic overview of the systematic method. The desire that makes the processor execute more data in a shorter time interval is always the first priority of our designers. We here propose to accelerate the software application by using hardware acceleration theory. We will emulate the software application by substituting the software part with the corresponding hardware part, so the data flow will automatically neglect the software function but go through the hardware accelerator. Hardware accelerator will process the input data and send the result back to the CPU. The total processing time could be reduced if hardware accelerators can process the data at a faster speed. The system overhead could thus be overcome. System energy consumption and resource costs will also be considered. The trade-off of these three important indicators will be balanced and an optimum solution could be achieved.

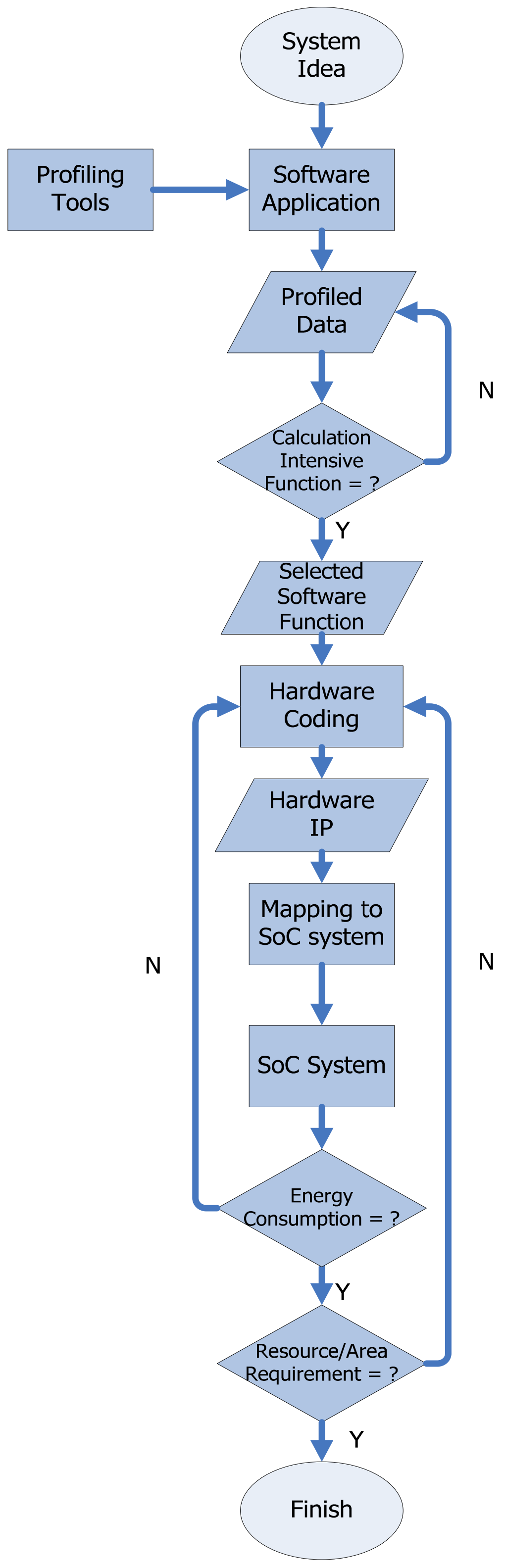


Figure III-3 Hardware Acceleration Method Framework

After the system concept is thrown out, software application will first be profiled by the software profiling method. Software profiling tools will be used to get the statistics of the software application. After the profiling process, profiling results will be analyzed and classified. The function, which will cost the most execution time and has very intensive calculation steps, will be chosen. Then, the selected function is transformed to the corresponding hardware using hardware description language. Xilinx Virtex 5 FPGA board will be used as the SoC platform [90]. The Xilinx MicroBlaze soft processor will be used as our on-chip CPU [67].

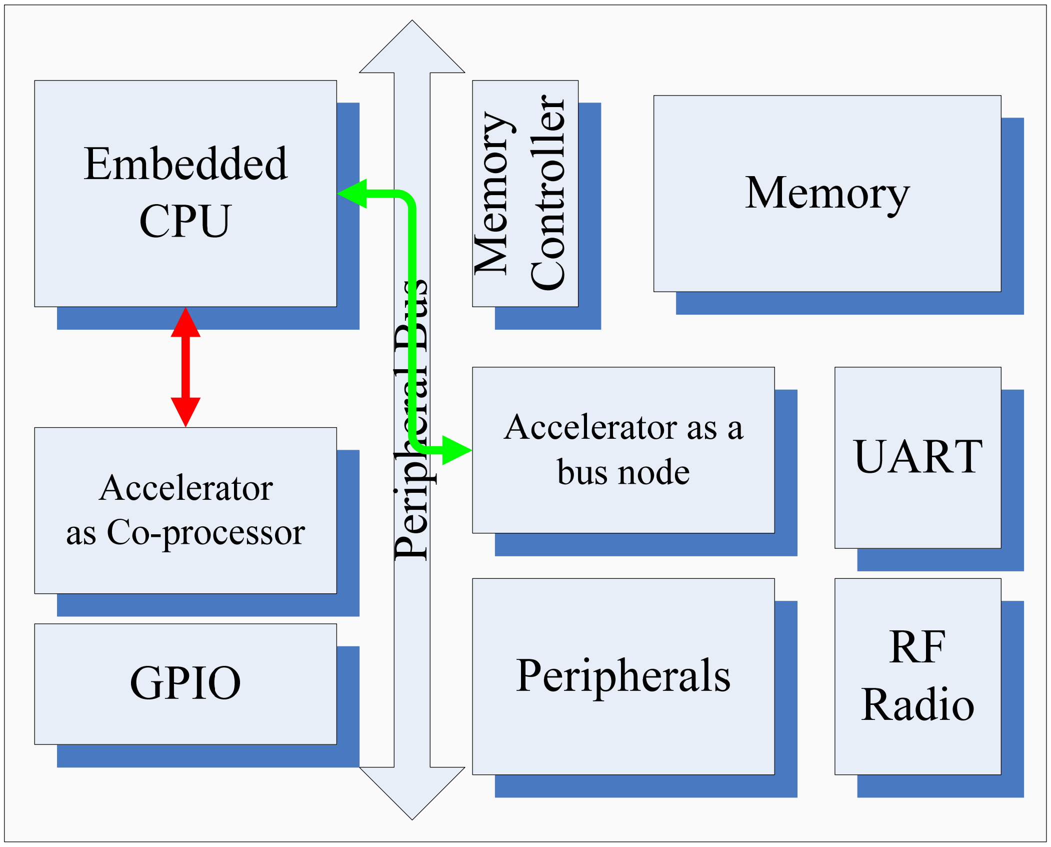


Figure III-4 Proposed Acceleration Method

In this system, the hardware part will be wrapped as an IP, which is connected to the system bus or directly connected to MicroBlaze's high speed link. As shown in

Figure III-4, the green line connects designed IP to MicroBlaze through the system bus, while the red line directly connects the designed IP to MicroBlaze. Each method has its own advantage, disadvantage, and different configuration with the embedded CPU. The software source code will be executed in the MicroBlaze but the hotspot function will be replaced by the hardware IP. By providing the driver, which will lead the data stream into the IP, and send the result back to the processor from hardware IP, the total execution time will be reduced greatly.

Speed, energy consumption and resource costs are three major concerns for modern system developments. Designers must balance the trade-off of these three factors. If the design has very high speed but at the same time it will cost huge energy, which goes beyond our system requirements, it cannot be viewed as a very good design. After the installation of the customized IP, energy consumption of the whole system with and without the IP will be compared. In our method, the system with the IP will consume more power than the system without the IP, since the extra IP will cost more transistors. If the system with the customized IP spends much less time than the system without the customized IP when executing the same amount of data, the total energy consumption will be reduced after the system runs longer enough. The longer the software application runs, the more energy is saved. Usually, the energy consumption will have a threshold depending on the requirement of the system standard. If the energy consumption could not meet the requirement, then the whole hardware IP should be optimized to fulfill the task, until the energy consumption meets the requirement. In this scenario, the designer must pay attention to the hardware IP and use some optimization methods to lower the energy consumption.

Different coding concepts will result different hardware accelerators, and different hardware accelerators will consume different system resources. When designers are provided with enough system resources, they need to care only about the speed of software application. But when system resources are limited, designers must take resource costs of different hardware accelerators into considerations. Designers must balance the trade-off of system speed, energy consumption, and resource costs to find an optimum design.

3.1.5 Productivity and Scalability

The design in this dissertation uses a hardware-software co-design method. In this process, the hardware design productivity has been improved. Under this circumstance, hardware engineers can run a complete design flow to verify the system design. Commonly, hardware engineers use hardware design tools to optimize the system if they have time. Nowadays, our on-chip designs have become more and more complex, even though engineers can optimize parts of the system, they can hardly guarantee an optimal solution from a higher system level view. With the new method, engineers can get a complete working system in a short time. Then the designer can spend the remaining time for further optimization. The system could be extracted and analyzed from a higher level view. This will provide some thoughts, which may be overlooked from lower level view.

Thus, engineers can effectively use the remaining design cycle time by resolving the bottleneck of the system and provide a better system design.

From the software engineer side, first, they don't need to go through a new design curve. There is nearly no difference between writing code for the general purpose CPU and FPGA based embedded CPU. Secondly, the hardware could be generated by some automatic tools, the only thing they need is to use system C language to describe the functional level of hardware behaviors. This provides little stress on software engineers.

The new method has good scalabilities and adaption. The scalability usually means a SoC system can expand some of its components to bigger size. Usually this will squeeze more performance. For example, expand the cache size will absolutely increase CPU performance, the more processed data in cache, the better of the CPU performance in general. In our method, the hardware IP are not limited to one, or two, engineers can make as more IPs as they want if necessary.

3.2 System Software Profiling Platform

Over the past years, embedded SoC systems have grown rapidly due to the powerful processing abilities and the more integrated components. They are used nearly everywhere in our daily lives from personal portable devices to large industrial machines. However, this demand puts a lot of stress on embedded system designers. They must not only deal with more and more complex design processes, but also shorter design cycles to earn the profit. In order to quickly and efficiently manage the system, designers must get fully understand their embedded system and applications. They must make the application run smoothly on the system. If it couldn't meet the system requirements, designers must find the system overhead and improve it. The profiling tools are designed for this purpose.

If designers find the bottleneck of software application, they can choose two different ways to improve the code:

- Designers can use their software techniques to re-program the source code, which causes the bottleneck. For example, currently, most of the source code is coded sequentially. One way to overcome the inherent limitations of sequential programs is to program it using parallelism techniques. After that, the source code must be rebuilt and re-analyzed to check whether the reprogrammed code meets the requirements.

- If the bottleneck of software applications is identified, but it is not easy to be reprogrammed using the parallel techniques, or the target platform doesn't support parallelism at all, designers can convert the corresponding part to be a hardware accelerator to improve the performance. In this case, designers must pay attention to hardware constraints, like the chip resource, thermal dissipation, etc.

In order to efficiently design a hardware-software co-design system, designers must pay much attention on the hardware and software partition. A well balanced system depends on the effectiveness of the hardware-software partition. This is why profiling tools become so important because they determine which part of the application is the software bottleneck.

Profiling tools are Computer Aided Design (CAD) tools, which could be used to evaluate the performance of a system. It is based on the time needed to finish certain functions in software applications. They can also be used to detect communication bottlenecks of a system. Profiling tools could be used by embedded SoC system engineers to guarantee their designs meets the system requirements [50, 51].

3.2.1 Existing Profiling Tools

As stated in chapter 2.2.2, based on implementation strategies, software profiling is classified into three methods: Software Based Profiling (SBP), Hardware Based Profiling (HBP), and FPGA based profiling. Each method has its own advantages and disadvantages. There are some typical profiling tools.

gprof is a software based profiling tool. It is an open-source profiling tool, which is more suitable to profile general computing software than embedded softwares [55]. It is mainly used by Linux and UNIX workstations to profile C/C++ code. Basically, it looks into application functions and inserts instrumentation code into a binary executable file to collect the timing information. The instrumentation code generates interrupts to sample the Program Counter (PC), so it can get the number of called functions by the sample frequency.

gprof will cause inaccuracy because the profiled time depends on the sampling frequency. Besides, inserting the instrumentation code will result in a number of interrupts in the software application, which may lead to unpredictable behaviors of applications running on the embedded SoC system.

For hardware based profiling tool. The Sun's UltraSPARC microprocessor has the hardware counters, which could be utilized by sun's software package [59]. The profiled information can be generated by detecting the hardware counter overflows. Hardware counters can monitor different types of events. For example: the instructions completed, the instruction cache miss, the data cache miss, etc. But these tools also have their drawbacks: First, they may not be able to execute an accurate timed trapping mechanism when a counter overflow occurs, so the value of the counter may not be accurate. Second, the backtracking mechanism used may not reach the desired result because some time it is difficult to find the address where an overflow occurred.

The Intel VTune Performance Analyzer is another type of hardware based profiling tool [91]. Intel Pentium microprocessors contain hardware counters, which could be accessed by the VTune performance analyzer. Designers can use this tool to identify and analyze the characteristics of software applications.

Figure III-5 is the workflow of the Intel VTune performance analyzer. In order to analyze the software application, designers must first build the original source code of the application. Then, after the correct configurations of the VTune performance analyzer, the executable file of the application will be run under the surveillance of performance analyzer. After the profiling process, the VTune performance analyzer will provide the detailed information of the software, including all the functions in the application and their corresponding time occupations. This information is very useful for further improvements of the application.

SnoopP is a real-time on-chip tool, which is used for profiling soft-core processors [50]. Numbers of counters are provided to users, they could be used to profile the performance of an application. SnoopP is non-intrusive to the code when profiling the application. But one deficiency is that the number of counters is restricted by the on-chip resources, the complexity of the whole system, and the number of functions user wants to profile.

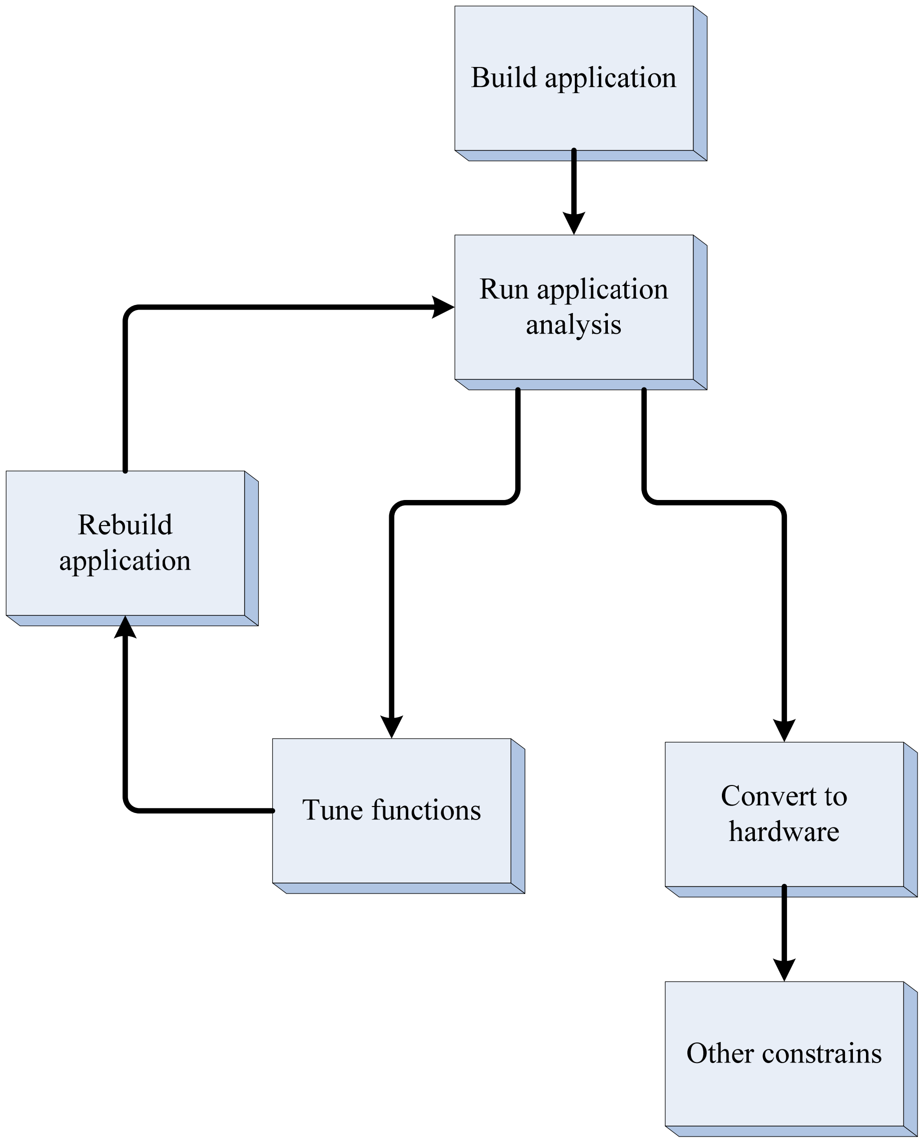


Figure III-5 Intel VTune Performance Analyzer Workflow

Table III-2 is the summarization of different profiling tools and their features. We use Intel VTune performance analyzer in our research project. Profiling tools are listed but not limited to these types.

Table III-2 Comparison of Profiling Tools

	SBP	HBP	FPGABP	Functional Profiler	Dynamic Profiling	Performance Overhead
gprof	●			●	●	●
VTune		●		●	●	
SnoopP			●	●	●	

3.2.2 Proposed Software Method

For an efficient and powerful SoC system, designers must fully understand their target hardware platform and the software application running on it. System specifications constrain the design method to a successful platform.

Currently, power consumption plays an important impact on the semiconductor industry, which is directly related to chip thermal problems and the energy consumption. In order to improve the software application performance and keep the energy consumption at an acceptable level, designers must find the bottleneck of software applications, which impacts the whole system performance. This bottleneck is called hotspot function.

The software program is executed by running different distinct functions. Normally, hotspot functions will consume a substantial amount of execution time for the specific algorithm. This will make them have a high probability to be used for runtime optimizations.

Designers can use different profiling tools to identify the hotspot function. Then the hotspot function will be transformed to the corresponding hardware and wrapped as an IP for system performance improvement. Traditionally, the function, which costs most execution time, is usually used for hardware acceleration.

However, this may not true for large software designs. The common hardware-software co-design method is specified to the specific software application. For large hardware-software co-designs, even the most experienced designer may not be able to take a comprehensive consideration. A universal method should be proposed to guide the process. As for the software partition, even though more execution time can be a critical factor for hotspot function, other concerns also need to be considered.

After taking a deep look at the application and its internal functions, an attribute called cycle per loop is analyzed and viewed as an important factor. As shown in Figure III-6, suppose the function has only one loop, the lower red arrow means the function need only a few cycles to finish the execution. In this way, the function is pretty easy for the CPU to execute. They may not have intensive computational steps, and there may not be significant differences, if the software function is transformed to the corresponding hardware. Thus, this function is not suitable for hardware acceleration. Even though the function is transformed to hardware IP, and the function executes a lot of rounds and costs lots of execution time, the CPU may not gain performance improvement. On the other side, if the function costs a lot of cycles to finish execution, then the function may not be able to be transformed to the hardware easily. Moreover, what if this function only executes few times in software application? It may be a waste of hardware resources, especially if the accelerated function only runs a few times.

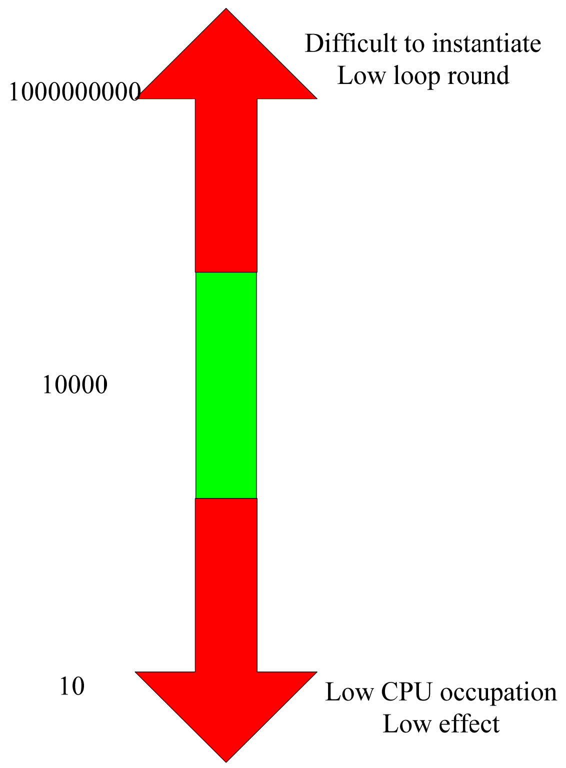


Figure III-6 Cycle Per Loop Analysis

So from this way, execution time shouldn't be the only aspect to consider in profiling methods. Instead, two additional attributions called cycle per loop and loop round are also important for the software profiling method.

As shown in Figure III-7, the x-axis stands for the loop round of the software application, while the y-axis represents cycles per loop of the software application. Points on the figure stand for the corresponding software functions. Through the figure, it is shown that the point B is a function, which has the lowest priority to be transformed to hardware IP. It has low cycles per loop and low loop rounds, so the execution time must be the lowest. When analyzing the software behaviors, it is obvious that some of the functions are not suitable to be converted to hardware. For example, the intensive data accessing functions. Hardware developments may not have significant improvement for these kinds of functions, because the majority of the execution time of these functions is waiting time for data access. Search window in H.264 is this type of function. Suppose a file is needed from the other part of the world, much time will be cost on request and acknowledgement stages. These kinds of functions are located in the area where point A is in the figure. Another concern is high loop rounds and low cycles per loop scenario, like the single thread functions, which are not easy to be accelerated by parallelism technology. For example, a function just reverses the value of a signal. Even if it could be looped for millions of rounds and has a long execution time, it is not suitable to be accelerated because there is little difference between the software and hardware approaches. These kinds of function are located around the area where point C is.

So, obviously, the target hotspot functions should not only have longer execution time but also higher loop rounds and cycles per loop. These functions are located around the area D in the figure. Functions located in area D should have higher priority to be viewed as candidates for the hardware acceleration. Transformed functions will greatly reduce the burden for the embedded processor and improve the system performance.

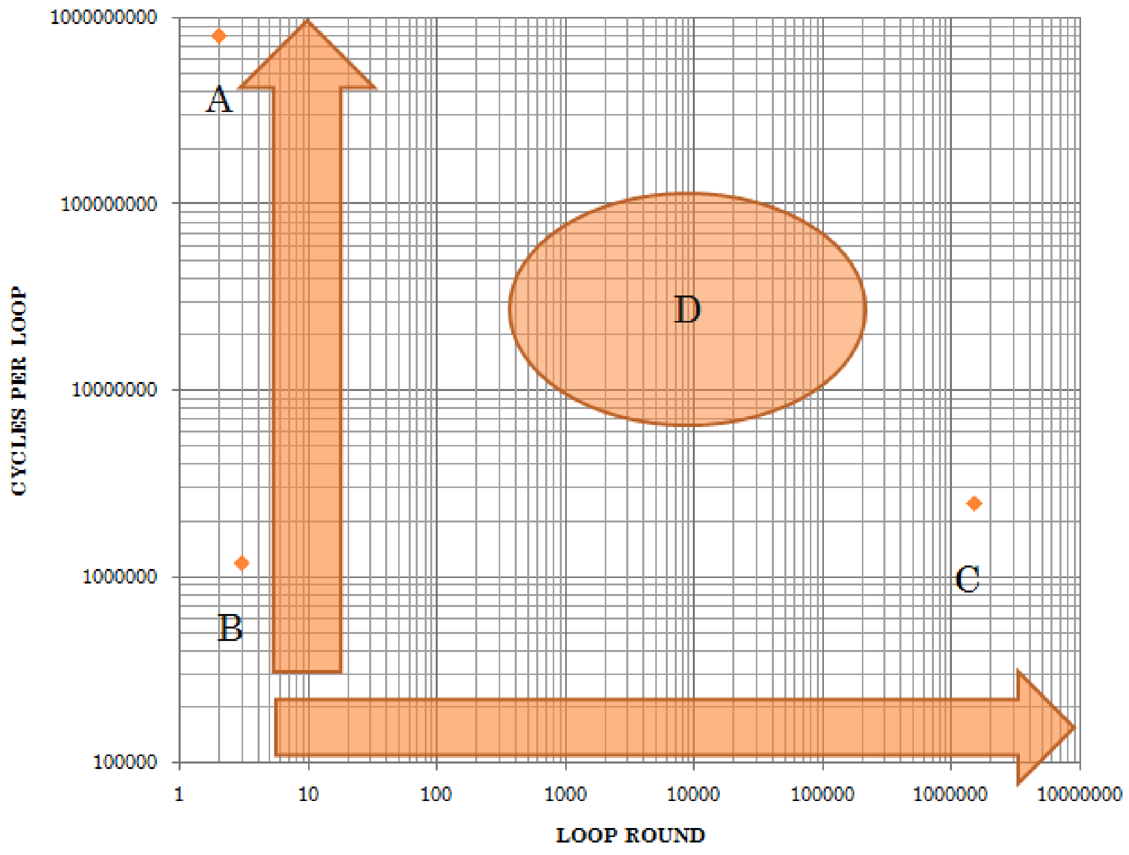


Figure III-7 Proposed Profiling Theory

3.3 System Hardware Platform

3.3.1 Design Hardware Accelerators

The hardware acceleration has the ability to speed up specific operations. This method allows the processor and its attached accelerator to run at the same time. In our research, the CPU will assign specific functions to the attached accelerator, while executing its own instructions at the same time.

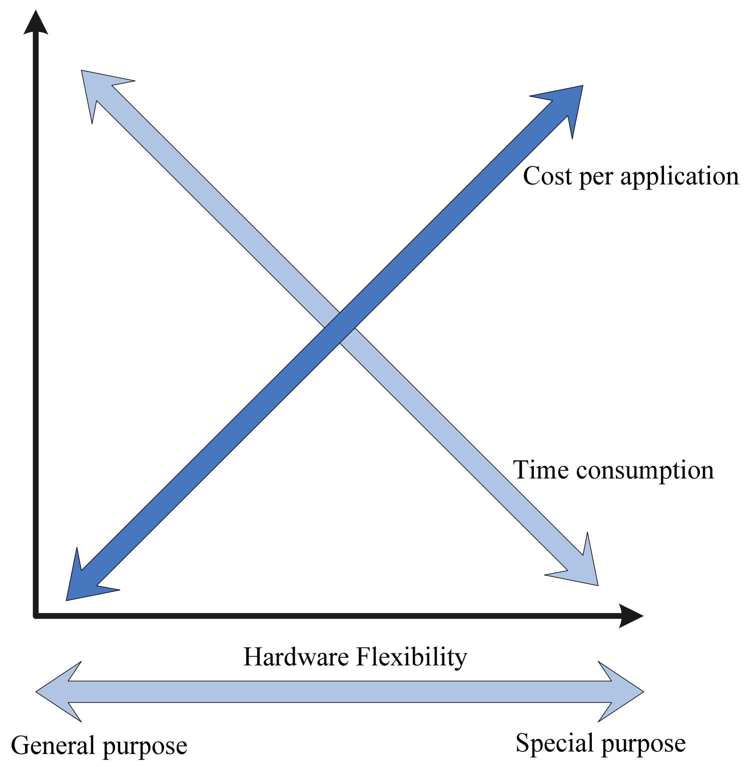


Figure III-8 Hardware Implementation Trade-off

The hardware acceleration method is very efficient to accelerate the desired functions, and resolve the bottleneck of software applications. However, there is a trade-off among performance improvement, power consumption and on-chip resource costs. Designers must use their techniques to balance these three critical indicators based on the system requirements.

Hardware accelerators have been used in CAD for many years. Their usages give us a lot of performance improvements in computer system design. However, hardware implementation also has the trade-off.

Figure III-8 shows the hardware implementation trade-off. In our daily lives, different types of hardware implementations are used. From one hand, the majority of the hardware products, which are widely used in the world, are general purposed products; on the other hand, special purpose hardware products are used for some special functions. They have different features such as:

- Flexibility. The general purpose hardware must be designed to fit as many applications as possible. The special purpose hardware is designed only to fit some limited functions. So general purpose hardware has the most flexibility. They could be modified based on the system requirement. While special purpose hardware has the least flexibility, which is not easy to be modified.

- Processing time. Because the general purpose hardware is designed to run a bunch of applications, it must include as many requirements as possible. The speed of the software application running on general purpose hardware is relatively slow. If the general purpose hardware is optimized for some special function, it may not able to deal with another function smoothly or directly stop working. But the special purpose hardware is designed for only specific functions. It could use as many optimization methods as possible to shorten processing time.

- Cost per application. The development of hardware will go through a number of processes. Suppose a general purpose hardware cost the same as the special purpose hardware. Since the general purpose hardware can execute a lot of applications, while the special purpose hardware can only execute specialized functions. The costs per application of the general purpose hardware will absolutely be lower.

If people design a hardware accelerator, they must also think about the trade-off of the overall system. Besides, other requirements like power consumption and chip resource costs must also be considered.

Three different hardware acceleration methods are listed in chapter 2.3. Each of them has its own advantages but they also have their own disadvantages:

- For the method using new created instruction. It is difficult to realize it in our research project. The major difficulty is that in order to compile a new instruction, the compiler must be modified to recognize this group of instructions. Another difficulty is that the number of chosen instructions should be confined to a limited size to avoid long internal latency. If one instruction executes many steps in the processor, other applications or functions will be interrupted, so the CPU efficiency will drop to a low level.

- For the co-processor design method. It is very efficient when the data transmission between the IP and processor is not very huge. When the data to be processed by the IP exceed the capacity of the processor's cache capacity, the data will be fetched from the off-chip memory and sent through processor to the dedicated IP. This will have great impacts for the processor, because the processor could not process as many requests as usual, thus the efficiency is very low. Besides, the total number of the attached IPs is limited by the processor's available high speed interfaces. The processor's resources may not meet the IP's demands.

- For the Bus-IP design method. All the communications between different devices are went through the system bus, so the performance improvement will be greatly affected by the speed of the system bus and the bus protocol. When a lot of requests are made, the bus protocol will choose the highest priority request, which may not be hardware accelerator's request. The communication between the IP and the processor is not as fast as the co-processor method. Because whenever the processor sends a command to the dedicated IP, or the IP acknowledges the instruction from the processor, there will be a response time interval, which is caused by the processor and bus's own behavior. If many requests are made at the same time on the bus, the time interval will be even worse.

So, designers must fully understand their design and evaluate different acceleration methods to make the right decisions.

3.3.2 Optimize the Designed Hardware

Designers can use different languages to design the desired hardware. The most prevailing language is the Hardware Description Language (HDL). They can also use a high level language like system C to describe the function behavior, and then use some CAD tools to convert the system C language to hardware directly.

In our research, Verilog HDL language is used to design the hardware and verify it. Verilog HDL is suitable for different levels of descriptions (as shown in Table III-3). Because Verilog HDL is an industry standard, designs using Verilog HDL could be easily mapped to different chips from different manufacturers.

Table III-3 Description form Different Level

	Behavior Description	Structural Description
System level	System algorithm	System logic diagram
RTL level	Data flow chart, FSM	Register, ALU, ROM
Gate level	Boolean equation, truth table	Logic gate, Flip-Flop
Schematic level	Geometrical figure	Figure connection

When the hardware design is finished, engineers must re-check their designs to see whether there has the chance to optimize them. There are many optimization method which could be used. Here are typical examples, which are used in our research.

Suppose the original design is to add four one bit data together.

$$Y = a + b + c + d \qquad \text{III (1)}$$

This will cost three one bit adders as shown in Figure III-9. But actually, this design is not an optimized design especially consider it will consume too much time. The design will need at least three clock cycles to finish the calculation process. Data is added sequentially, this is the slowest design.

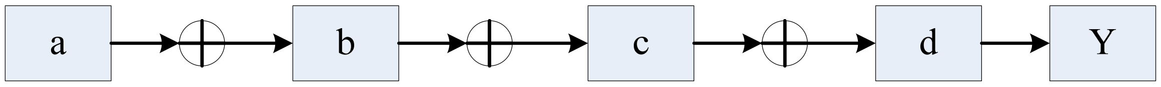


Figure III-9 Original Adder Design

Then this design is optimized, two operands are grouped together to make the data flow run faster.

$$Y = (a + b) + (c + d) \quad \text{III (2)}$$

The optimized design is shown in Figure III-10. Even it still uses three adders, two groups operands are calculated at the same time, it will only need two clock cycles to finish the calculation. The hardware design frequency is increased by using this design.

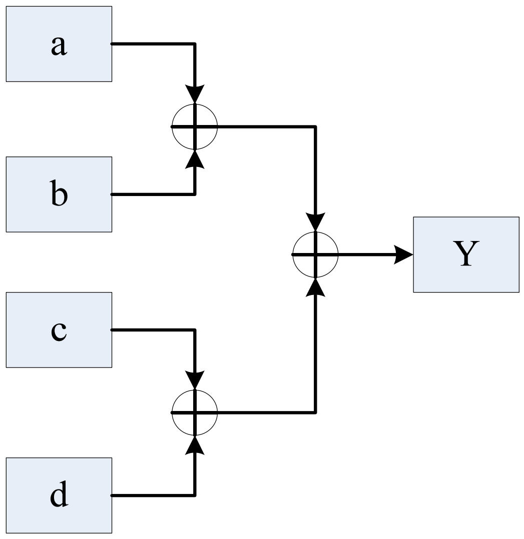


Figure III-10 Optimized Design I

But this is still not fully optimized if designers still want the hardware to run at a higher frequency. The improved design is shown in Figure III-11. It uses the pipeline technique to improve the processing time. In the first three cycles, pipeline technique processes the data in a normal way. After that, this hardware design can generate the result every clock cycle. So in this way, the hardware operation frequency is increased, thus the processing performance is improved.

One thing needed to know is that the figure is a simplified flow chart. If designers use pipelines in their hardware design, there should have registers, which are used to store the processed value. The data will be processed and stored every pipeline stage.

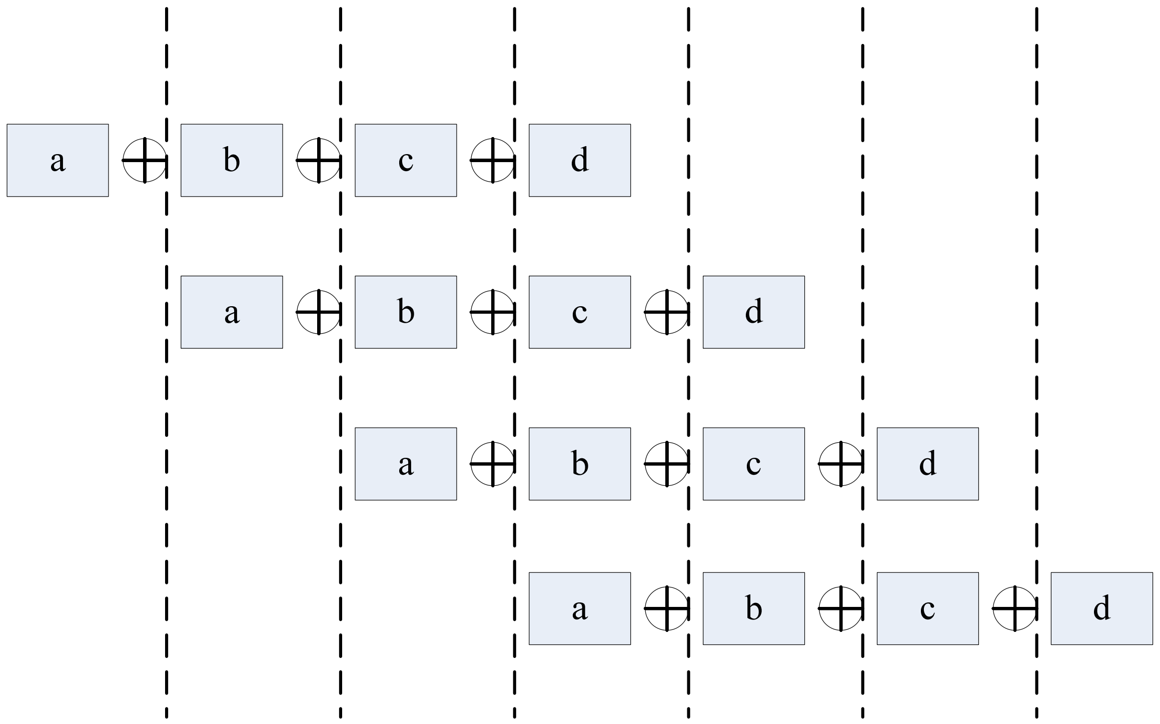


Figure III-11 Optimized Design II

3.4 Summary

In this chapter, the system design method is introduced. A detailed introduction about the two important parts, which involves the software profiling method and the hardware acceleration method, is described.

For the system design method, it starts from the SoC top-down design workflow, and the hardware-software co-design idea. After that, a overall workflow of the design project is presented. The method includes the software part and hardware part. With the design method, the SoC design can have a better productivity compared to the traditional design flow.

The software profiling platform is used to help the designer fully understand the software applications. Applications running on the SoC system may have bottlenecks, which will cause a great impact on system performance. Designers can use the software profiling method to identify the hotspot function, which causes the system overhead. Then, further optimizations could be made.

After identifying the hotspot function in software applications, designers can use the hardware acceleration method to accelerate the aimed function. This will not only improve the execution time of the system, but also reduce the energy consumption.

Hardware designs always have trade-offs. Designers must balance different aspects of the system, especially system speedup, energy consumption, and resource costs. Different system designs will have different specifications. So it is important for designers to have a full view of their design.

IV SYSTEM SOFTWARE PLATFORM EXPERIMENTS AND RESULTS

4.1 Video CODEC Acceleration Development

H.264 is the most commonly used video compression, especially for high resolution, low distortion, and low bit rate, which are the standard settings for a video. However, compressing the HD video format (720p and 1080p) into a portable format (H.264 or MPEG4) is very time consuming, e.g. Real-time encoding using H.264 requires up to 3600 Giga Instruction per Second (GIPS) and 5570 Giga Bytes per Second (GBPS) according to HDTV720p requirement.

Even though the Real-Time encoding is not necessary for most of the cases, but when needed, the video compressing processes can take up to 20 hours in order to compress a 30 minutes HD video. Nowadays, average electronic consumer uses HD video to record moments of their daily lives such as TV shows, or precious events and even everyday occurrences so that they can either share or watch at a later time. It then becomes a necessity for the computer architecture to include a hardware module that can facilitate or accelerate the video coding/encoding process.

It is necessary to find out hotspot functions of H.264 video CODEC in order to accelerate the processing time. In [92, 93, 94], the authors found the hotspot functions based on the execution time. In [95, 96], the authors said the hotspot functions could be identified by monitoring the loop rounds. However, their methods are not accurate enough to identify the hotspot functions of the H.264.

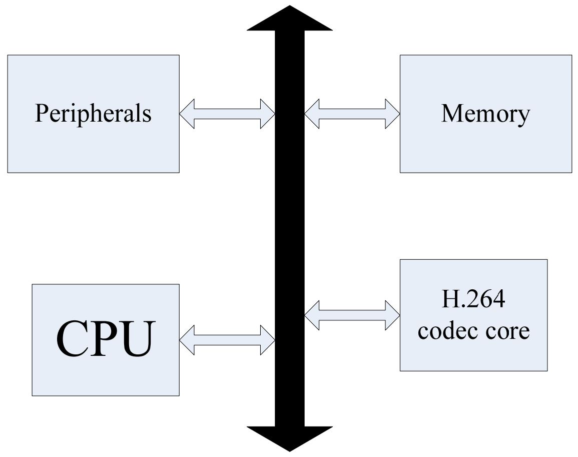


Figure IV-1 General H.264 based SoC Design

The execution time and energy consumption are two key performance indicators for modern system development. An efficient design is comprised of a system that can process a lot of operations while minimizing its energy consumption. A more general architecture to accelerate H.264 CODEC video standard is shown in Figure IV-1. However, designing a specific model with H.264 processing ability involves a general knowledge of Application-Specific Integrated Circuit (ASIC) design, which can be a time consuming process. Furthermore, the hardware that makes up the H.264 block will require a significant number of on-chip transistors, which in some cases would be very inefficient especially if on-chip hardware resource is limited. A general hardware assisted approach was presented by Chen *et al*. [97]. They used a pipeline methodology to accelerate the system performance, but their hardware accelerator was connected through the system bus. This seems to be an attractive process, but if many other hardware modules need to communicate with the processor at the same time, then the execution time can be affected. Moreover, they implemented the entire process in hardware instead of the hotspot function. In [98], Kordasiewicz *et al*. showed the speed and area optimizations for H.264 Discrete Cosine Transform (DCT) and quantization blocks, but for current embedded system development, the energy consumption is a very important factor to take into consideration. The DCT performs an extensive calculation to accomplish transformation, but it may not be the most suitable function to be accelerated in the H.264. Elgato [99] had a USB thumb disk as an H.264 hardware accelerator. Even though the USB 2.0 bus processes around 28MB/s, the Northbridge can take up to 2132MB/s, which is about 700 times faster than the USB bus (As shown in Figure IV-2). Not to mention that it will bring a lot of compatibility problems if we implement the chip into the Northbridge. H. C. Lin *et al.* [100] used DSP to accelerate the CODEC. However, the DSP is not as flexible as FPGA, and it is not inherently designed for parallel computing. R. Duart *et al*. [94] used Bus-IP design for acceleration. However, their design applied serial port to monitor the results, which would cause overhead too. Therefore, in this work, a hardware acceleration platform for H.264 encoder is implemented. After balancing the trade-offs among system performance, energy consumption and resource cost, optimum solution can be achieved.

4.2 Platform Setup Design

In this research, a new way to enhance the overall system performance and minimize energy consumption for the H.264 video CODEC is explored. H.264 is chosen to be the experimental target but not confined to it. The complete workflow includes the design process involving hardware and software co-design. This method expedites the implementation and evaluation of the proposed design including software and hardware validations.

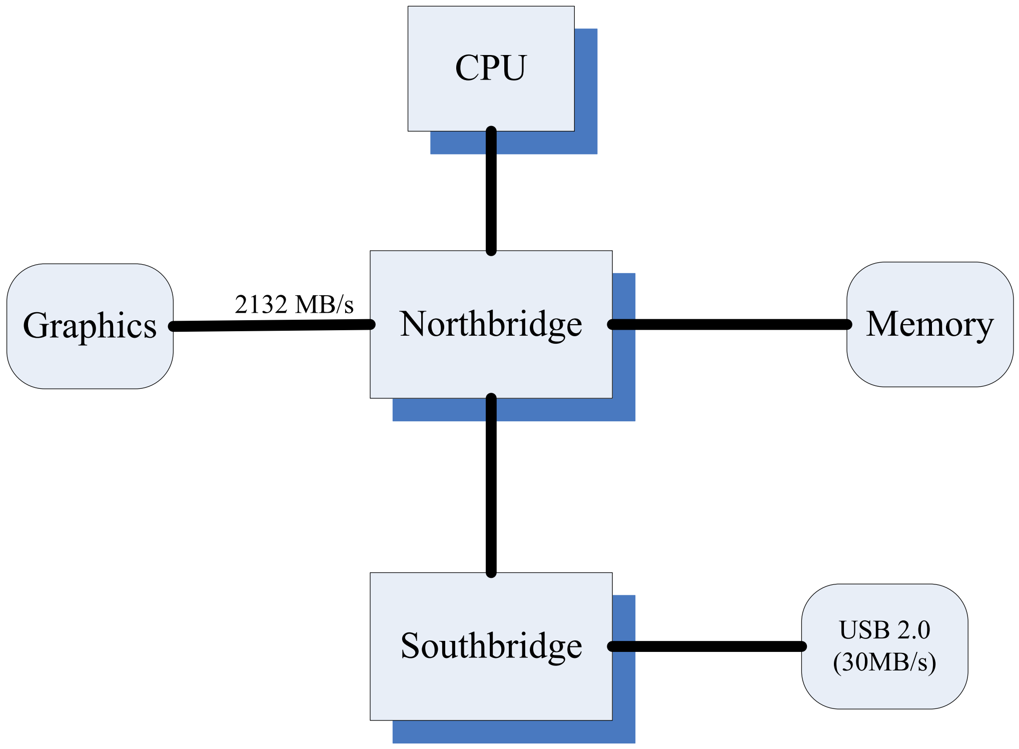


Figure IV-2 Data Speed of Computer Architecture

The process begins at the software implementation stage. The software, which needs to be accelerated, is written in a high level language such as C language. Then the software is verified and built into the executable file. The profiling tool is used to get the detailed information of the software executable file to determine which part of the code is considered as the hotspot, so that it can be implemented as a hardware accelerator. The hotspot function consumes most of the execution time as well as energy consumption. Hence, the designer needs to carefully analyze the statistics of the software and determine the hotspot function. If a software implementation does not meet the system performance constraints, the designer needs to use the hardware acceleration method to optimize the code by moving the costly hotspot function into a hardware platform as a hardware accelerator. The corresponding hardware accelerator must use less execution time and energy consumption. Therefore, the overall system performance is improved and energy consumption is reduced. If this is not the case, then, the designer must go through the complete system design flow again until the modified SoC system meets system requirements.

4.2.1 H.264 Software Module

The H.264/AVC standardization has two conceptual layers as shown in Figure IV-3. These two layers need not only to be efficient, but also to be compatible with all current and future protocols and network environments. The Video Coding Layer (VCL) efficiently defines the representation of the source content of the video encoding. The Network Abstraction Layer (NAL) formats the representations from the VCL and provides header information to enable the output data to be suitable for a variety of media formats. The following lists the features of H.264 [101]:

- Multi-picture inter-picture prediction
- Flexible interlaced-scan video coding
- Spatial prediction for intra coding
- Intra coding directional spatial prediction
- Arithmetic entropy coding
- In-loop de-blocking filtering
- Lossless macro-block coding

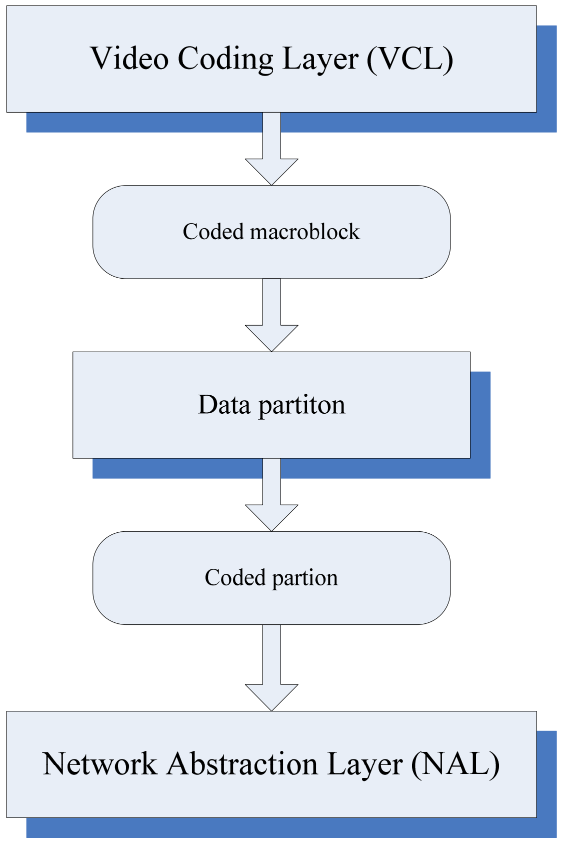


Figure IV-3 Structure of H.264/AVC Video Encoder

- Flexible slice size
- Small block-size transform
- Weight prediction
- Arithmetic entropy coding

4.2.2 Video Compressing Process

The official standard C model of H.264 (called “JM”) is used to setup the H.264 environment. The following provide detailed information on how it works:

1) Prepare a sequence of un-compressed YUV frames.

Table IV-1 Consecutive Experiment Frames

Frame Seq. #1	Frame Seq. #2
Frame Seq. #3	Frame Seq. #4
Frame Seq. #5	Frame Seq. #6

YUV is a color space which is different from the additive color model called RGB. YUV is used as a color space standard by Phase Alternation Line (PAL), National Television System Committee (NTSC) and Sequentiel Couleur Avec Memoire or Sequential Color with Memory (SECAM). YUV standard allows reduced bandwidth of chrominance components; it typically enables transmission errors or compression artifacts to be efficiently masked by the human perception [102].

YUV color space model defines the color in terms of three components, the Y stands for luminance, which is the brightness. The U and V stand for the chrominance information. In the past, the pure black-and-white system used only Y information. The U and V information were added separately so that a black-and-white system can still receive and display color information.

Table IV-1 shows six consecutive frames captured from HD video. Our experiment will be executed based on these frames.

2) Prepare a configuration file that contains the right format and information of the YUV frames. The configuration file includes the essential constraints of the compressed method, such as frame rate, source specification, encoder control, rate control, etc.

3) Build H.264 encoder to be an executable file from JM source code. The executable file is built by the C language compiler.

4) Use a profiling tool such as Intel VTune performance analyzer to monitor the execution of the software. Then, the software statistics can be gathered through run-time profiling. Profiled data need to be analyzed and if not satisfied with the outcomes, further process needs to be done until the desired value is obtained.

4.2.3 Experimental Results and Analysis

In this experiment, The Intel processor, which allows Intel VTune performance analyzer to use its hardware counters for profiling, is used. Because using hardware based profiling tool does not introduce any instrumentation code, the source code of the compiled application remains intact. Furthermore, profiling tool gathers the performance data during the runtime of software application, so it does not add performance overhead.

Table IV-2 shows the profiled data collected by the VTune performance analyzer, VTune provides a lot more information about each function running in the software application. In this research, three factors will be focused: Loop rounds, Cycles per loop, and Execution time.

After profiling the H.264 compressing process using VTune, and as shown in Table IV-2, HadamardSAD transformation takes most of the overall execution time, following the SetupFastFullPelSearch and iabs functions. From the table, the number of cycles per loop of SetupFastFullPelSearch can also be obtained. Even though, this function has the highest number of cycles per loop, it only executes relatively small loop rounds. Meanwhile, iabs has the highest number of loop rounds, but its numbers of cycles per loop are low. As for Hadamard, it cost more loop rounds than SetupFastFullPelSearch function and much more cycles per loop compared to iabs function. Thus Hadamard function is the best choice for acceleration.

Table IV-2 Profiled Data for H.264

	Loop rounds	Cycles per loop	Total clock cycles (x1000)	Ratio (%)
HadamardSAD4x4	13094634	6413.437	83981615	39.18
SetupFastFullPelSearch	7560	6789512.169	51328712	23.95
iabs	193135005	186.695	36057255	16.82
FastFullPelBlockMotionSearch	309960	107192.415	33225361	15.50
SetupLargerBlocks	7560	538094.974	4067998	1.90
quant_4x4_around	709201	450.025	319158	0.15
writeUVLC2buffer	4179607	67.876	283696	0.13
generateChromaXX	588	340318.027	200107	0.09
forward4x4	878713	172.957	151980	0.07
getNonAffNeighbour	3308413	44.406	146914	0.07
memcpy	194945	675.657	131716	0.06
writeCoeff4x4_CAVLC_normal	352007	316.619	111452	0.05
residual_transform_quant_luma_4x4	711522	128.159	91188	0.04
sample_reconstruct	351336	216.289	75990	0.04
GetMotionVectorPredictorNormal	436339	139.323	60792	0.03
inverse4x4	414672	140.494	58259	0.03
SubPartitionMotionSearch	40320	1444.916	58259	0.03
submacroblock_mode_decision_p_slice	10080	5779.663	58259	0.03
predict_nnz	575038	96.908	55726	0.03
BlockMotionSearch	309960	179.784	55726	0.03
mode_decision_for_I4x4_blocks_JM_High	48384	1151.744	55726	0.03
quant_ac4x4_around	177408	299.834	53193	0.02
compute_residue	188496	255.321	48127	0.02
get4x4Neighbour	3045736	14.97	45594	0.02
compute_SSE4x4	429282	100.309	43061	0.02

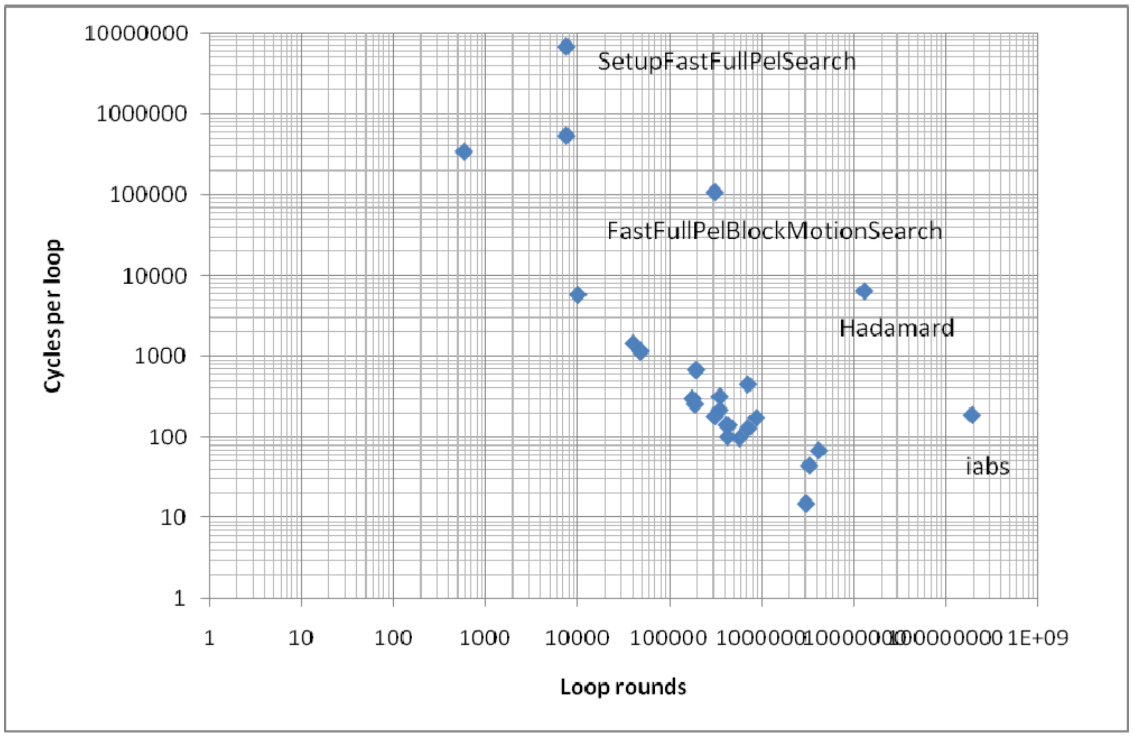


Figure IV-4 Statistics of H.264 Video CODEC Encoding

Figure IV-4 is the dotted graph for profiled H.264 information. The position of each function is shown in the figure based on loop rounds and how many cycles each loop takes. As mentioned in Figure III-7, a function, which has high number of cycles per loop, loop rounds as well as execution time, should be best suitable for the hardware acceleration. Even though, functions, like iabs, have high execution time and loop rounds, it only accounts for a small number of cycles per loop. In this case, it may not be the best suitable for acceleration. Moreover, this function only gets the absolute value of the input data. If this function is implemented as the hardware IP, it will not have significant performance improvement because the feature of the function itself. Hence, it is not considered as a good candidate for the hardware acceleration. On the other hand, the SetupFastFullPelSearch function has higher cycles per loop number, but this function is not used as often as other two functions. Besides, converting a searching function to hardware will not benefit the system performance a lot; it is not a calculation intensive function. Thus, it cannot be considered the best candidate for acceleration. The Hadamard transformation has intensive data calculation, higher cycles per loop, and loop rounds in the H.264 video coding process. With higher execution time, it is the best candidate to be accelerated. Thus, it is the main focus for further optimization.

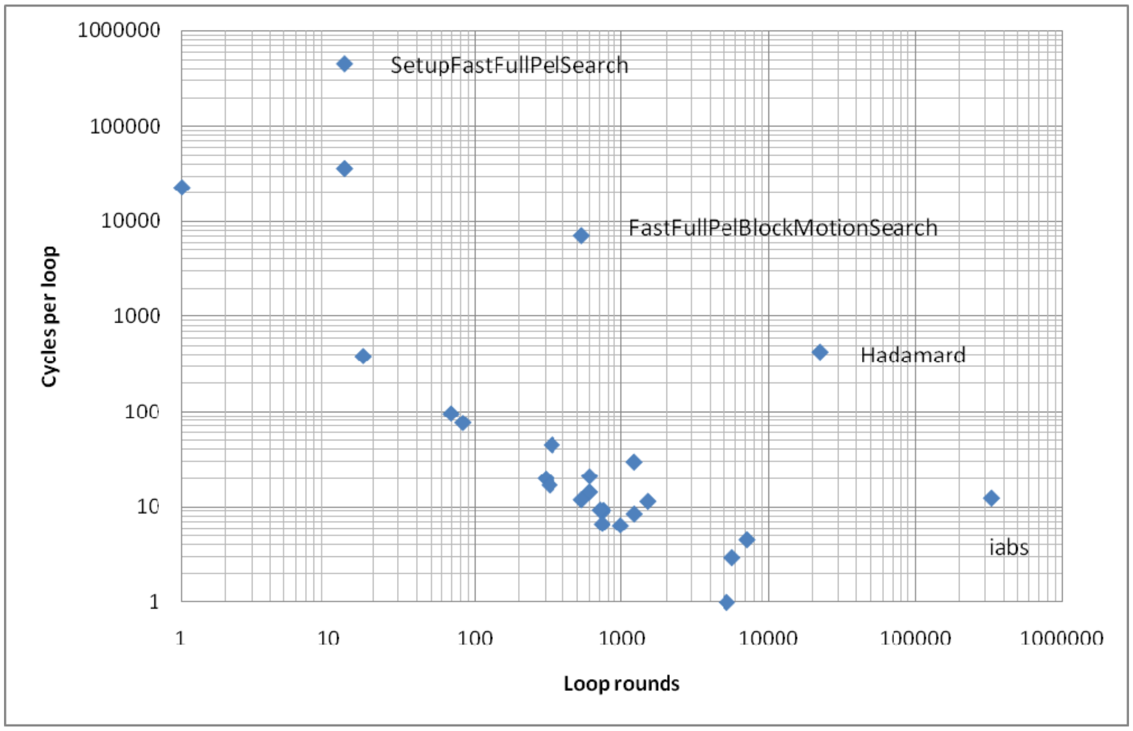


Figure IV-5 Normalized Statistics of H.264 Video CODEC Encoding

Figure IV-5 is the normalized graph of the profiled data. It clearly shows that the HadamardSAD transformation function is located in the desired area which we mentioned before. It has higher priority to be accelerated using the hardware acceleration method.

4.3 Summary

In this chapter, the well known H.264 module is used to validate the software method for further hardware acceleration.

H.264 is used as the target software application, but the software profiling method is not limited to the H.264 CODEC. The software identification method can be used to identify the hotspot function of the software application. Conventionally, if software application does not meet the system requirement, the designer needs to choose the function, which has the highest execution time for further optimization. If there is no profiling tool to be used, designers need to choose the function based on their experience. From our research observation, a real application cannot be accelerated only based on the result of execution time. Even an experienced engineer may not be able to balance the trade-off of performance and energy on a complex and large system design.

Thus, the software identification method is validated in this chapter. Besides of the execution time, two key factors called cycled per loop and loop roundd are introduced for internal functions of the software application. Hardware counters in the Intel processor could be used by the hardware profiling tool. After analyzing the profiled data, it is clear that Hadamard function is the identified hotspot function in the H.264 video CODEC standard, and it is the aimed function, which is suitable for further acceleration.

V SYSTEM HARDWARE PLATFORM EXPERIMENTS AND RESULTS

This research is to explore a new design workflow, which can help engineers efficiently design a balanced SoC system. Therefore, after profiling the H.264 software application and identifying the hotspot function, the FPGA platform is used to accelerate the software application and verify the acceleration method.

5.1 FPGA Hardware Platform

The FPGA is an attractive alternative for rapid prototyping of large embedded systems. Due to its reconfigurability and flexibility, designers can use abundant on-board resources for fast design verification, while keeping the design costs at a low level.

The hardware acceleration, which allows the system to execute concurrently and achieve performance improvement, can make specific operations run at a higher speed. The processor directs the data flow to the hardware accelerator, while executing its own instructions at the same time. However, people need to understand that even the hardware accelerator can execute specific functions as fast as possible, it doesn't mean that the more hardware parts, the better of the performance. Adding more hardware parts to the system means more power consumption. Obviously, there is a trade-off between performance improvement and power consumption.

This research project uses a Virtex 5 FPGA board as the hardware platform. The reconfigurable ability of FPGA makes us easily customize the hardware implementation without implementing the whole hardware into a real physical chip. Therefore, the workload, which involves hardware-software co-design, can be modified and evaluated rapidly.

In the past, even though transistors on a single FPGA chip increased enormously, designers may still face the problem that the on-chip resources are not enough. Thus, a good design is a system, which can execute applications fast enough, consume less power, and occupy less on-chip resources.

In this research, instead of entirely implementing the aimed application, we selectively implement the chosen hotspot function of a software application. The hardware accelerator is used as a customized IP, which can decrease the computation time. Performance improvement can be observed when the application is running on the SoC system. Meanwhile, the trade-off among performance, system energy consumption, and resource costs are balanced.

5.2 Identified Hotspot Function

5.2.1 Hadamard Transformation

The software profiling method is used to profile the H.264 software application. The Hadamard transformation is identified as the hotspot function after analyzing the profiled data. Then, the hardware description language is used to design a customized Hadamard IP, which is connected to the hardware platform. Different hardware coding styles will result different types of customized IPs. Designed hardware IPs are evaluated based on the system speed, resource costs and energy consumption. A better solution can be chosen after balancing these three aspects.

The Hadamard transformation is a generalized Fourier transforms. It performs symmetric, involution, linear operations on 2^m Real numbers. The Hadamard transform H^m is a $2^m \times 2^m$ Matrix, which is a square array of plus and minus ones, whose rows are orthogonal to others. If H is a $N \times N$ Hadamard matrix, then the product of H and its transpose is an identity matrix. A Hadamard matrix can only exist for n is 1, 2, or multiple of 4. The matrix I is an identity matrix [103, 104, 105].

$$H_n H_n^T = nI_n \qquad \text{V (1)}$$

From properties of the Hadamard matrix, it is shown that if H is an N order Hadamard matrix, then the 2n order of Hadamard matrix is:

$$H_{2n} = \begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix} \qquad \text{V (2)}$$

A two order Hadamard matrix is:

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad \text{V (3)}$$

From equation V (2), the four order Hadamard matrix can be generated:

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \qquad \text{V (4)}$$

Normally, the Hadamard transformation consumes a lot of computational steps. The naive implementation will have a computational complexity of O(N^2). Thus, the fast

Hadamard transformation is developed to decrease the computational complexity. A fast Hadamard transformation requires only NlogN additions or subtractions. It recursively breaks down an original Hadamard transform of size N into two smaller transforms of size N/2. Figure V-1 is the 8 vectors fast Hadamard transformation.

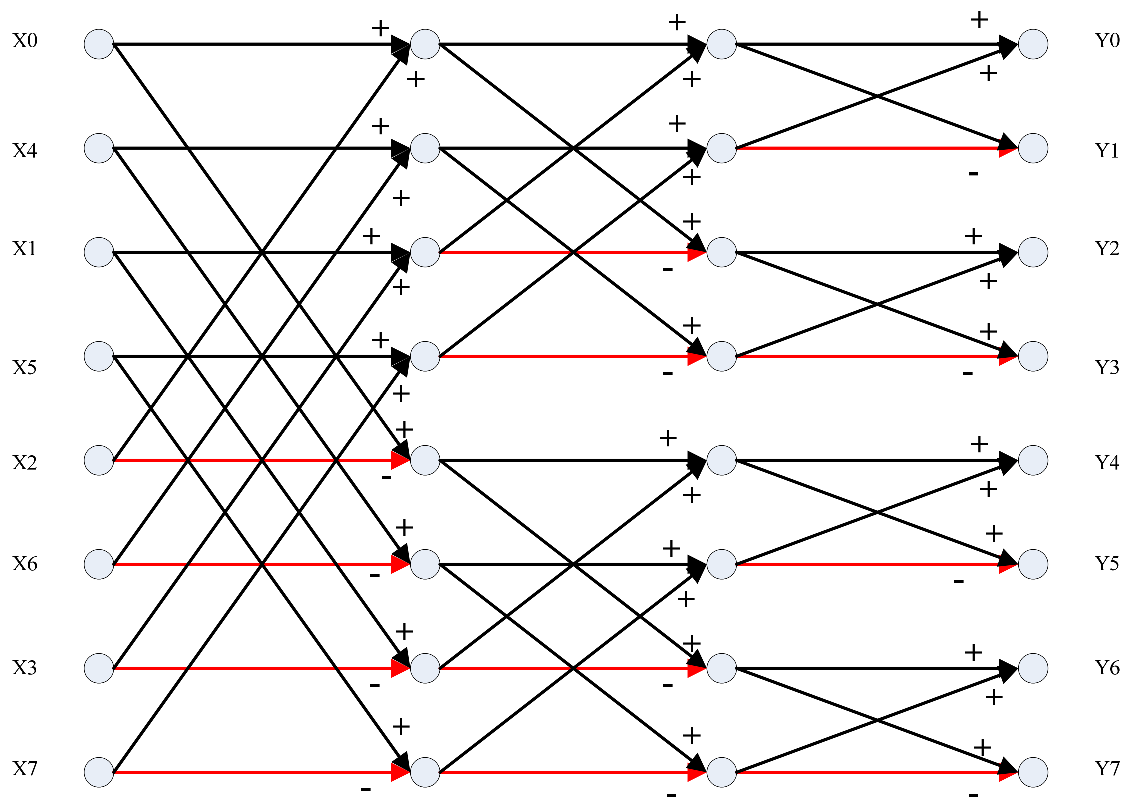


Figure V-1 Fast Hadamard Transformation of Eight Vectors

In our research project, the 4 × 4 orders Hadamard matrix is identified as the hotspot function in the H.264 video coding standard. It plays an important role in the whole H.264 coding process.

5.2.2 Hardware Design of Hotspot Function

In the proposed hardware design method, the trade-off of system performance, hardware cost, and energy consumption are balanced to get the optimized design. In order to find the optimum solution for this research project, five typical kinds of Hadamard accelerators are designed and mapped to the hardware experimental platform. Different performance data are evaluated based on the Virtex 5 FPGA board [106].

Five types of Hadamard IPs are designed and mapped to the hardware platform, so that specific performance data can be gathered. The designed hardware accelerators are:

- The original Hadamard transformation using wire connections, which directly connect different signals together to the output. Named Hadamard_1.
- Different from the original way, each of the Hadamard calculation result is stored in the register to get the output value. Named Hadamard_2.
- The fast Hadamard transformation is used for the designed hardware accelerator. The Hadamard transformation here has three layers, and signals are directly connected together to the output. Named Hadamard_3.
- For the fast Hadamard transformation, registers are used to store the results of only one layer. Others are still connected by direct wire connections. Named simplified pipeline Hadamard_4.

- For the three-layer fast Hadamard transformation, registers are used to temporarily store calculation results of different layers. Named three-layer pipeline Hadamard_5.

A synthesis tool, which is Synplify pro [107], is used to get the detailed hardware resource consumption of these five designs. Then, Hadamard IPs are mapped to the designated FPGA board in order to get a fully understanding of the system.

The resource costs of five Hadamard designs are listed in Table V-1. It is shown that the Hadamard_2 costs the most on-chip LUTs of the designated FPGA chip. The Hadamard_2 costs 1462 LUTs, and then following the Hadamard_1, which costs 1334 LUTs. The LUT costs of other designs have little difference. The Hadamard_2, Hadamard_4, and Hadamard_5 use different numbers of on-chip D-type Flip-Flops. The Hadamard_2 costs the most, while the Hadamard_4 costs the least. The maximum frequency of each design is shown in the table also. The gate level map of each design is listed in Appendices.

Table V-1 Resource Usage of Hadamard Design

	Hadamard_1	Hadamard_2	Hadamard_3	Hadamard_4	Hadamard_5
Max Freq	64 MHz	512 MHz	162 MHz	293 MHz	492 MHz
FD	N/A	1379	N/A	128	447
MUXCY_L	749	1211	440	440	441
XORCY	841	1211	441	441	441
LUT	1334	1462	529	530	504

Combined with the future experimental results, such as performance improvement and energy consumption, an optimized design can be found from this research project.

5.3 System Architecture

The FPGA is famous for its powerfully reconfigurable and programmable ability. So there is no need to build the real hardware chip to verify the functionality of designs. The Virtex 5 FPGA board is used to implement our proposed system. It is the most affordable FPGA platform and widely used by academia. The Xilinx MicroBlaze is used as the on-chip soft-core, which connects the Hadamard IP through different ways to get the performance data.

The system architecture is shown in Figure V-2. The Xilinx Virtex 5 FPGA board with a MicroBlaze CPU is used as the SoC platform. The MicroBlaze is a 32bit Reduced Instruction Set Computing (RISC) architecture. It can be configured to different frequencies based on different designs. In our SoC system, the CPU uses 50MHz frequency to guarantee every Hadamard IP works smoothly. The MicroBlaze has its Local Memory Block Random Access Memory (BRAM), which can be configured at maximum 64KB. The BRAM can be accessed through the Local Memory Bus (LMB) by two separate buses, one is Instruction LMB (ILMB) [108], and the other is Data LMB (DLMB). Both ILMB and DLMB are 32bit buses and can be accessed by the MicroBlaze in a single clock cycle. If one SoC system doesn't have off-chip external memory, all the instructions and data will be stored in the BRAM and executed by the processor.

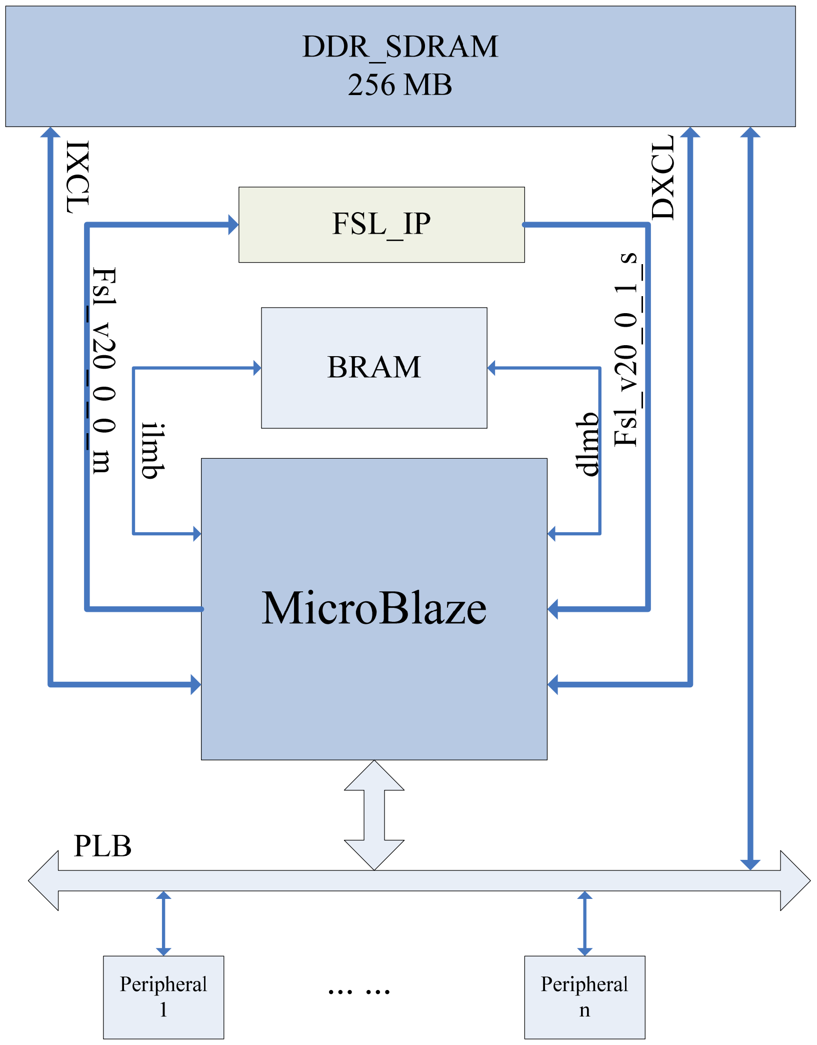


Figure V-2 Basic MicroBlaze based SoC System

There is an on-chip system bus called Processor Local Bus (PLB) [109] for the MicroBlaze to communicate with other peripherals, such as DSP IP, serial connection IP, and other customized IPs. The PLB bus has its own protocol, which manages the data transmission. This MicroBlaze-based platform is a memory-mapped SoC system. Each peripheral or embedded IP will have a specific memory address range and can be accessed only by providing its corresponding memory address. Therefore, if the MicroBlaze wants to communicate with a peripheral, it must provide exact address.

The SoC system uses the PLB bus, which is relatively slow, for communication among different components. However, if a lot of data and instructions need to be transferred, the data congestion will happen and system performance will be degraded. Thus, Xilinx provides another connection method, called Fast Simplex Link (FSL) [110]. The FSL is a single-directional point-to-point interface. The MicroBlaze can have up to 16 pairs of FSLs. Each pair of the FSL has one input and one output channel. The data, which will be processed, can be transmitted between the MicroBlaze and customized IPs immediately.

Instructions and data can be stored in the off-chip memory besides the BRAM, called DDR_SDRAM. It is 256MB on our board. The MicroBlaze can access this off-chip memory through the PLB bus or the Xilinx Cache Links (XCL). One XCL pair includes one Instruction XCL (IXCL) and one Data XCL (DXCL).

5.4 System Bus-IP Design

5.4.1 Experiment Setup

As stated before, there is one type of hardware acceleration method called Bus-IP design. In this design, the CPU sends the data needed to be processed to the hardware accelerator, which is connected to the system bus. The hardware accelerator processes the data and then sends results back to the CPU. The designed hardware is good at processing computational intensive and parallel functions. Thus, if the aimed hotspot function meets our software identification standards mentioned before, performance improvement can be achieved.

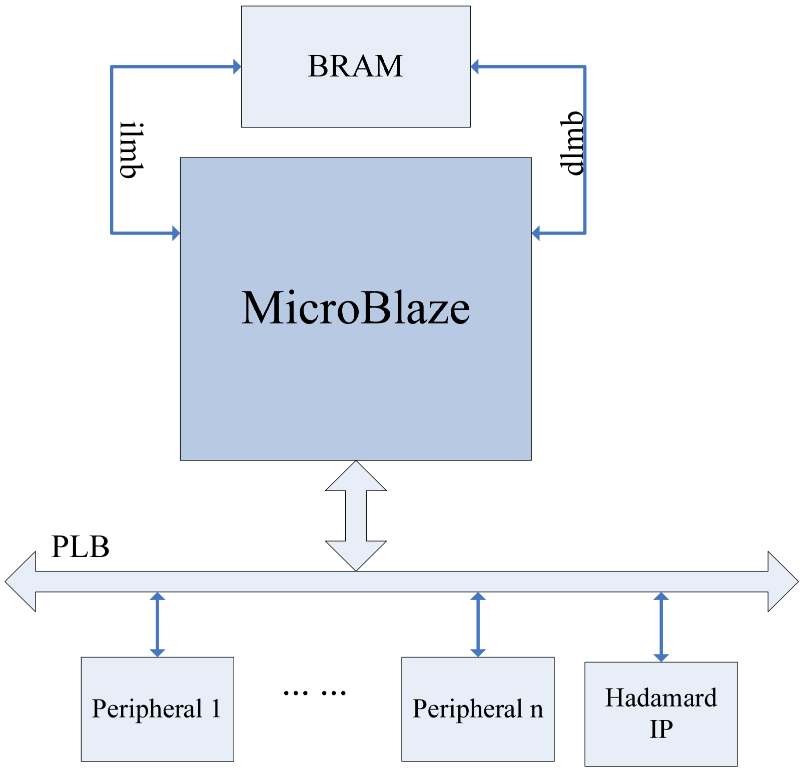


Figure V-3 Bus-IP Design for Hadamard Acceleration

The system Bus-IP design being used is shown in Figure V-3. Instructions and data, which will be processed, are stored in the BRAM. When accessing the data and instructions, the MicroBlaze provides an exact address of the BRAM. This system uses the PLB bus as the system bus. The MicroBlaze communicates with the PLB bus through specific internal ports. The PLB can be connected to different types of peripherals, such as DSP, wireless IP, etc. The designed Hadamard IP, which has its own address, is connected to the PLB bus. When the MicroBlaze wants to communicate with the Hadamard IP, it first points to the address where the Hadamard IP is. Then, the driver in the MicroBlaze guides data from the BRAM through the PLB bus, and finally reaches the hardware IP. After finishing the processing procedure, the Hadamard IP sends results back to the MicroBlaze through the PLB bus. The data, which flow on the PLB bus, must obey the PLB bus protocol. Under this way, The MicroBlaze first sends a request to the PLB bus. If the bandwidth on the PLB bus is enough, the arbiter on the PLB bus acknowledges the request, and transfers the data to the desired block, which is the Hadamard IP. When the data need to be sent back, the PLB also acknowledges the request from the IP.

Based on software profiling results before, the Hadamard function is the hotspot of the H.264 software application. If the system can process the data using the hardware IP instead of the pure software method, performance gain can be improved. Five types of Hadamard IPs are implemented to this Bus-IP design. Performance and energy consumption data are gathered.

The Xilinx XPower analyzer is used to estimate the power consumption of the system [111]. The XPower analyzer is a Xilinx tool used for analyzing the power consumption for a post implemented routed design. It collects information about the aimed hardware and provides an accurate estimation about the power consumption. The overall power consumption of the hardware design can be gathered by using the XPower analyzer. As mentioned before, the hardware design costs different amounts of resources. Compared to the pure software design, the hardware method uses more on-chip resources, even though the hardware spends less time than the software. This means the hardware design has higher power consumption than the pure software method. On the other hand, if the hardware can greatly reduce the software execution time, and if the software application runs long enough, the energy consumption can be reduced. Energy consumption is a key aspect of the modern SoC design. Therefore, with the help of the XPower analyzer, the energy consumption of the proposed system can be generated.

5.4.2 Experimental Results

Table V-2 is the system performance and energy consumption comparison of the Hadamard_1. By providing different amount of processing data, the CPU clock cycle costs can be generated. The Modelsim [112] is used to simulate the running system. Then, the speedup of the system performance can be generated. The energy consumption of the pure software method and the hardware acceleration method are compared, so that the saved energy consumption can be collected.

Table V-2 Hadamard_1 PLB Performance and Energy Consumption Comparison

Input Size	Software Cycle	Hardware Cycle	Speedup	Software Energy Consumption (μJoules)	Hardware Energy Consumption (μJoules)	Saved Energy Ratio (%)
16	372	132	2.82	7.232	4.873	32.62
160	3720	1319	2.82	72.317	48.733	32.61
1.6 K	37199	13201	2.82	723.165	487.374	32.61
16 K	371999	131999	2.82	7231.673	4873.434	32.61

As the data set increases, it is shown from the table that the hardware costs less time. The average speedup for the Hadamard_1 is 2.82X. The energy consumption is reduced by using the hardware IP. The average saved energy consumption is 32.61%.

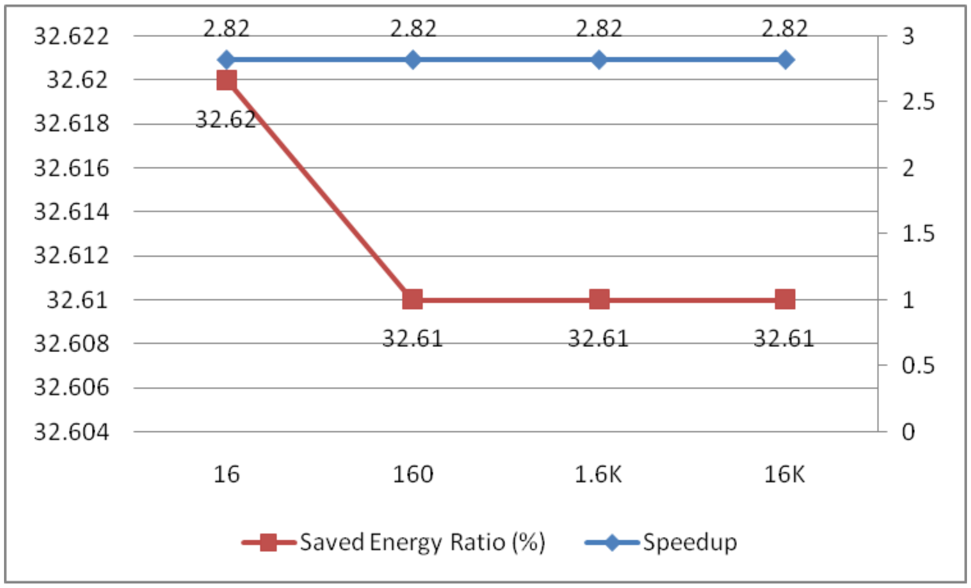


Figure V-4 Bus-IP Design Data for Hadamard_1 Accelerator

Figure V-4 shows the speedup and saved energy ratio for the Hadamard_1. The Hadamard_1 uses wires to connect inputs together, and the outputs are generated directly. So there is a constant speedup and saved energy ratio. Table V-3 is the system performance improvement and energy consumption comparison of the Hadamard_2.

Table V-3 Hadamard_2 PLB Performance and Energy Consumption Comparison

Input Size	Software Cycle	Hardware Cycle	Speedup	Software Energy Consumption (Joules)	Hardware Energy Consumption (Joules)	Saved Energy Ratio (%)
16	372	141	2.64	7.232	5.238	27.57
160	3720	1338	2.78	72.317	49.695	31.28
1.6 K	37199	13308	2.80	723.165	494.261	31.65
16 K	371999	133008	2.80	7231.673	4939.919	31.69

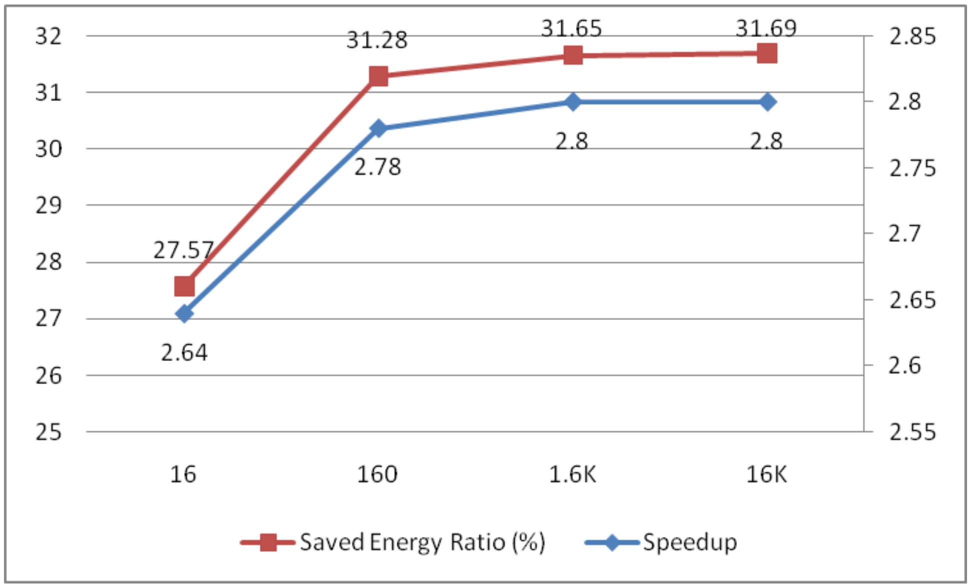


Figure V-5 Bus-IP Design Data for Hadamard_2 Accelerator

Figure V-5 is the speedup and saved energy ratio for the Hadamard_2. Results are generated each clock cycle except the first output data. The average speedup is 2.76X, and the average saved energy consumption is 30.54%. Table V-4 is the system performance improvement and energy consumption comparison of the Hadamard_3.

Table V-4 Hadamard_3 PLB Performance and Energy Consumption Comparison

Input Size	Software Cycle	Hardware Cycle	Speedup	Software Energy Consumption (Joules)	Hardware Energy Consumption (Joules)	Saved Energy Ratio (%)
16	372	132	2.82	7.232	4.873	32.62
160	3720	1319	2.82	72.317	48.733	32.61
1.6 K	37199	13201	2.82	723.165	487.374	32.61
16 K	371999	131999	2.82	7231.673	4873.434	32.61

The fast Hadamard transformation is used in the Hadamard_3 design. There are not many differences between the Hadamard_1 and the Hadamard_3, since wires are used to directly connect signals together. If signals are connected by wires, results can be generated immediately. There is no register in this design, so results are not impacted by the CPU. The average speedup for the Hadamard_3 is 2.82X, and the average saved energy consumption is 32.61%. However, the Hadamard_1 costs more on-chip resources than the Hadamard_3. Furthermore, the Hadamard_1 directly connects a lot of adders together. Each adder will cause timing latency; the overall timing latency will have great impact on the maximum frequency.

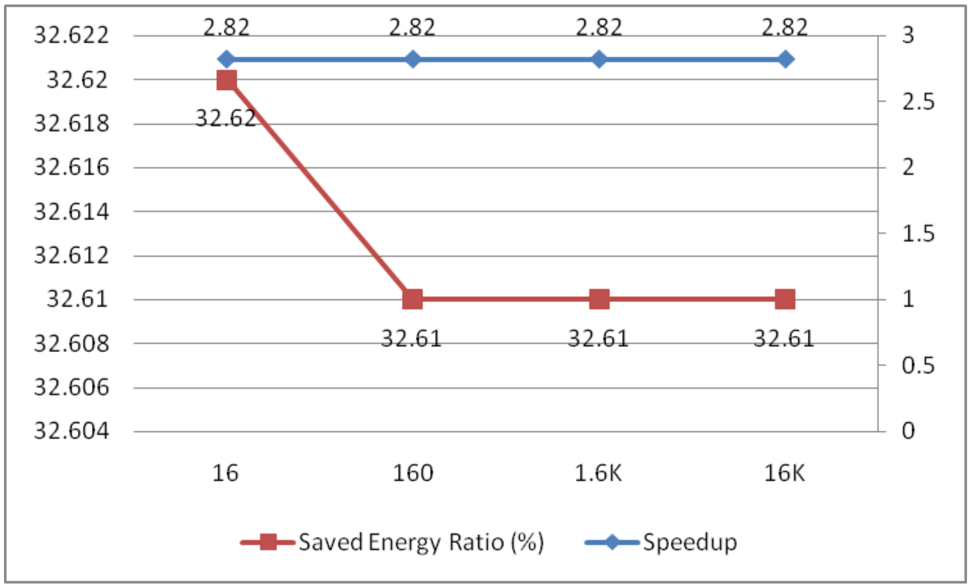


Figure V-6 Bus-IP Design Data for Hadamard_3 Accelerator

Figure V-6 is the speedup and saved energy ratio for the Hadamard_3. Using wire connections makes it have the same speedup and saved energy as the Hadamard_1 accelerator. Table V-5 is the system performance improvement and energy consumption comparison of the Hadamard_4.

Table V-5 Hadamard_4 PLB Performance and Energy Consumption Comparison

Input Size	Software Cycle	Hardware Cycle	Speedup	Software Energy Consumption (Joules)	Hardware Energy Consumption (Joules)	Saved Energy Ratio (%)
16	372	133	2.797	7.232	4.919	31.98
160	3720	1330	2.797	72.317	49.185	31.99
1.6 K	37199	13300	2.797	723.165	491.836	31.99
16 K	371999	1330000	2.797	7231.673	4918.342	31.99

Figure V-7 is the speedup and saved energy ratio for the Hadamard_4. The average speedup for the Hadamard_4 is 2.797X, and the average saved energy consumption is 31.99%. Table V-6 is the system performance improvement and energy consumption comparison of the Hadamard_5.

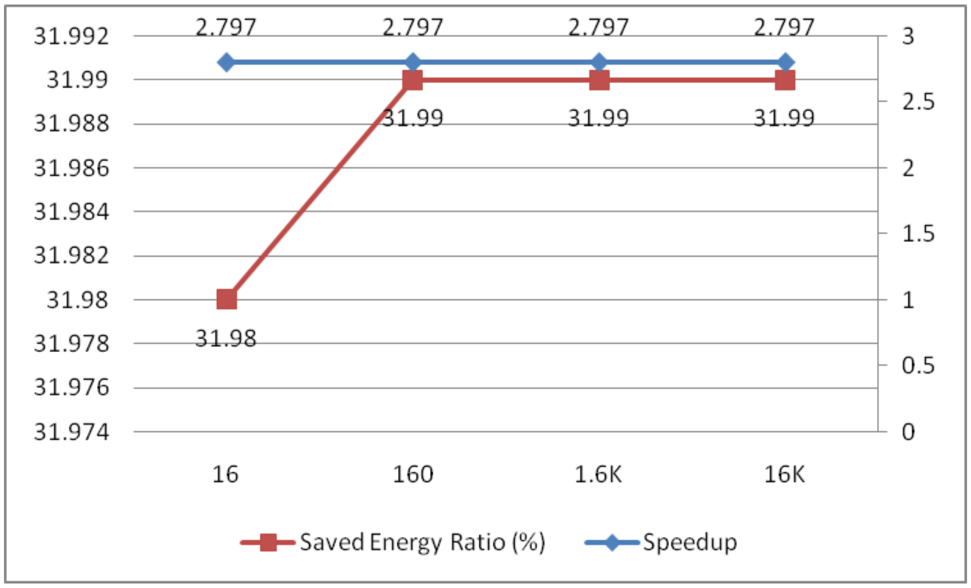


Figure V-7 Bus-IP Design Data for Hadamard_4 Accelerator

Table V-6 Hadamard_5 PLB Performance and Energy Consumption Comparison

Input Size	Software Cycle	Hardware Cycle	Speedup	Software Energy Consumption (Joules)	Hardware Energy Consumption (Joules)	Saved Energy Ratio (%)
16	372	136	2.735	7.232	5.040	30.31
160	3720	1333	2.791	72.317	49.382	31.71
1.6 K	37199	13301	2.796	723.165	492.669	31.87
16 K	371999	133003	2.796	7231.673	4926.440	31.88

Registers are used to store results of each layer, the first data set costs more time. After that, the output data will be generated every clock cycle. Figure V-8 is the speedup and saved energy ratio for the Hadamard_5. The average speedup for the Hadamard_5 is 2.780X, and the average saved energy consumption is 31.44%.

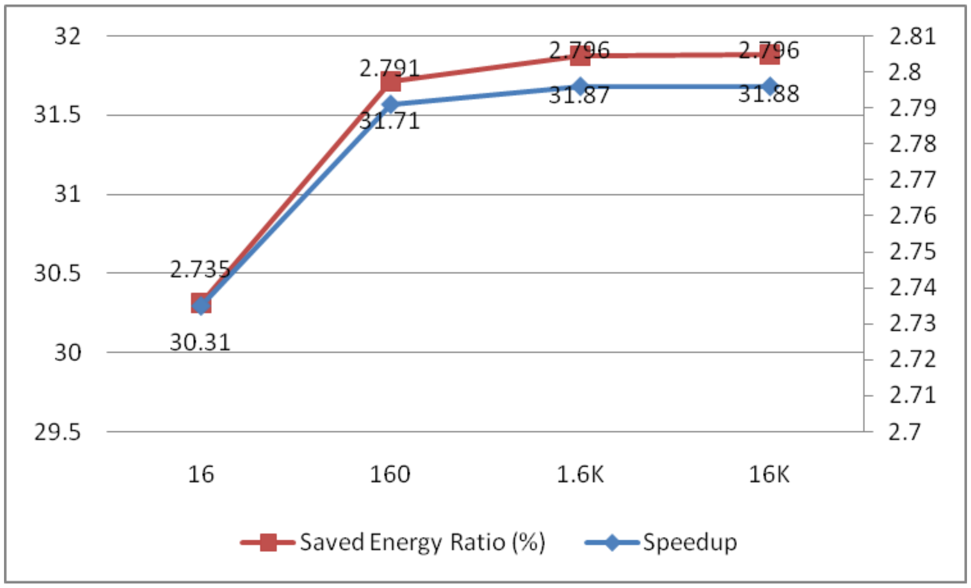


Figure V-8 Bus-IP Design Data for Hadamard_5 Accelerator

5.5 System Co-Processor Design

5.5.1 Experiment Setup

As mentioned before, the system Bus-IP design uses a system bus as a bridge for communication between the MicroBlaze and the Hadamard IP. However, this design has its own limitations. The system bus is a low speed bus, so if the system can use a higher speed link, it can have greater performance improvement.

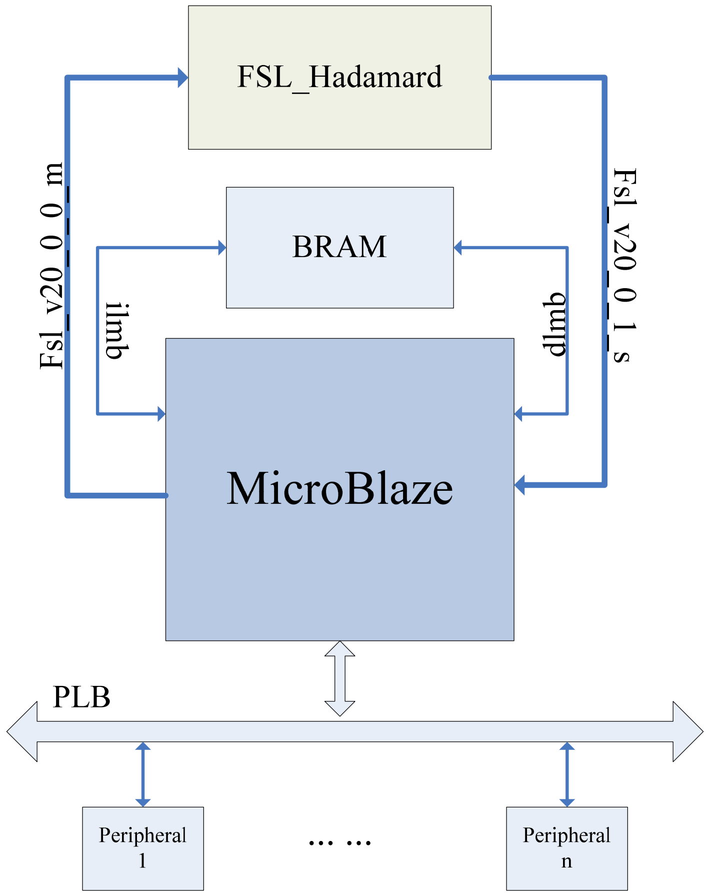


Figure V-9 Co-processor Design for Hadamard Acceleration

Figure V-9 is the co-processor design for the Hadamard acceleration [122]. Instead of using the PLB bus to connect Hadamard IPs to the MicroBlaze, the FSL is used as the communication link. In our system, the MicroBlaze is a 32bit soft processor running at a clock rate of 50 MHz. The MicroBlaze processor can communicate with the BRAM through the Instruction Local Memory Bus (ILMB) and the Data Local Memory Bus (DLMB). All the data and instructions are stored in the BRAM. The customized IP is directly connected to the Microblaze through the Xilinx FSL channel. The FSL is a high speed co-processor link. Each pair of FSL has one input and one output channel. The data, which will be processed, can be transmitted between the MicroBlaze and the customized IP at a faster speed.

5.5.2 Experimental Results

The software and hardware execution time can be gathered through experiments. The XPower performance analyzer is used to measure the energy consumption of the designed system. The Speedup and saved energy ratio can be generated from the measured data.

Table V-7 is the system performance improvement and energy consumption comparison of the Hadamard_1 using FSL. Since the FSL is a high speed link, the execution time of a function will be greatly reduced with the same hardware IP. The energy consumption will also be greatly reduced. The average speedup for the Hadamard_1 is 8.09X, and the average saved energy consumption is 76.52%.

Table V-7 Hadamard_1 FSL Performance and Energy Consumption Comparison

Input Size	Software Cycle	Hardware Cycle	Speedup	Software Energy Consumption (Joules)	Hardware Energy Consumption (Joules)	Saved Energy Ratio (%)
16	372	46	8.09	7.232	1.698	76.52
160	3720	460	8.09	72.317	16.984	76.51
1.6 K	37199	4600	8.09	723.165	169.833	76.52
16 K	371999	46000	8.09	7231.673	1698.334	76.52

Figure V-10 is the speedup and saved energy ratio for the Hadamard_1. Results are stable because the Hadamard_1 uses wire connections. Table V-8 is the system performance improvement and energy consumption comparison of the Hadamard_2.

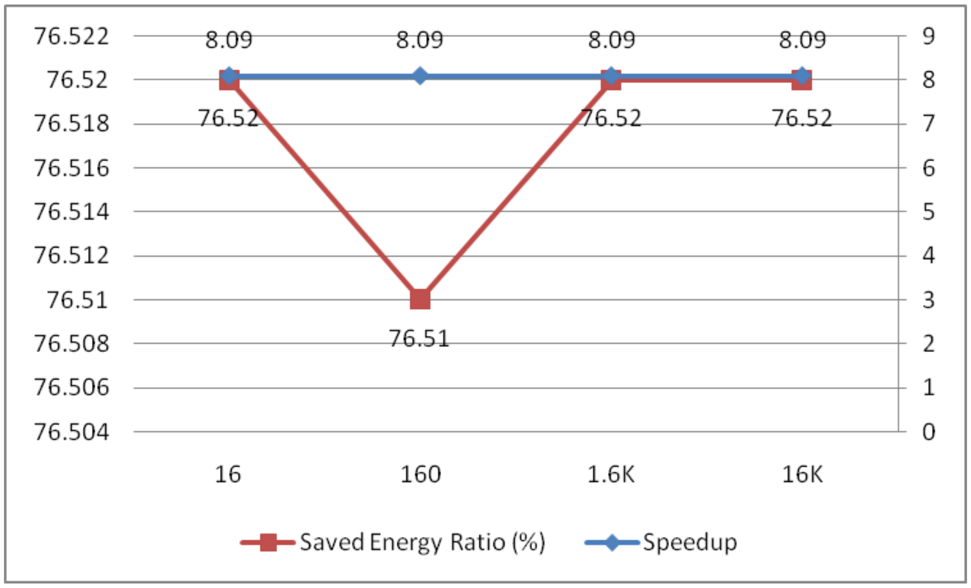


Figure V-10 Co-processor Design Data for Hadamard_1 Accelerator

Table V-8 Hadamard_2 FSL Performance and Energy Consumption Comparison

Input Size	Software Cycle	Hardware Cycle	Speedup	Software Energy Consumption (Joules)	Hardware Energy Consumption (Joules)	Saved Energy Ratio (%)
16	372	55	6.76	7.232	2.043	71.75
160	3720	478	7.78	72.317	17.755	75.45
1.6 K	37199	4708	7.901	723.165	174.857	75.82
16 K	371999	47008	7.914	7231.673	1745.892	75.86

Figure V-11 is the speedup and saved energy ratio for the Hadamard_2. The average speedup for the Hadamard_2 is 7.59X, and the average saved energy consumption is 74.72%. Table V-9 is the system performance improvement and energy consumption comparison of the Hadamard_3.

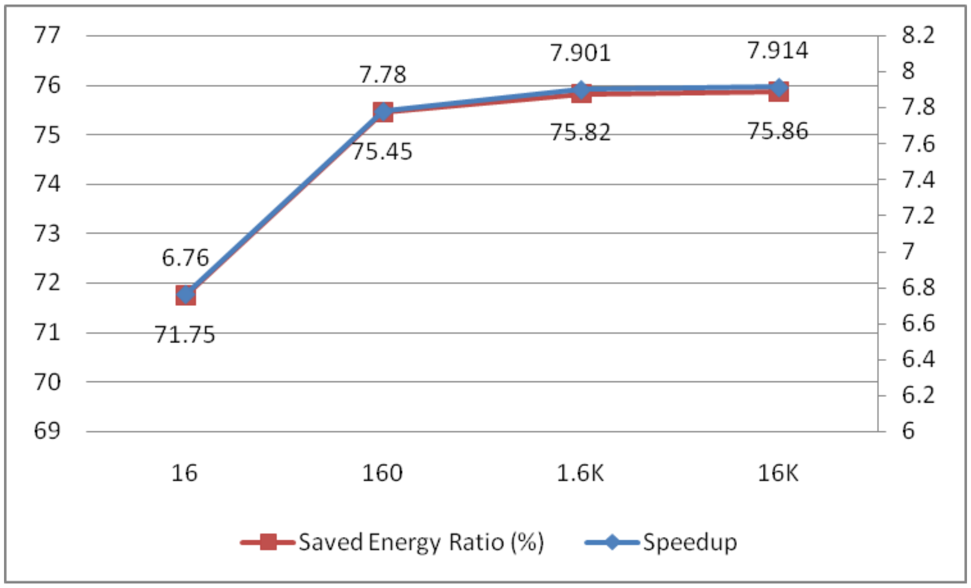


Figure V-11 Co-processor Design Data for Hadamard_2 Accelerator

Table V-9 Hadamard_3 FSL Performance and Energy Consumption Comparison

Input Size	Software Cycle	Hardware Cycle	Speedup	Software Energy Consumption (Joules)	Hardware Energy Consumption (Joules)	Saved Energy Ratio (%)
16	372	46	8.09	7.232	1.698	76.52
160	3720	460	8.09	72.317	16.984	76.51
1.6 K	37199	4600	8.09	723.165	169.833	76.52
16 K	371999	46000	8.09	7231.673	1698.334	76.52

Figure V-12 is the speedup and saved energy ratio for the Hadamard_3. The average speedup for the Hadamard_3 is 8.09X, and the average saved energy consumption is 76.52%. Table V-10 is the system performance improvement and energy consumption comparison of the Hadamard_4.

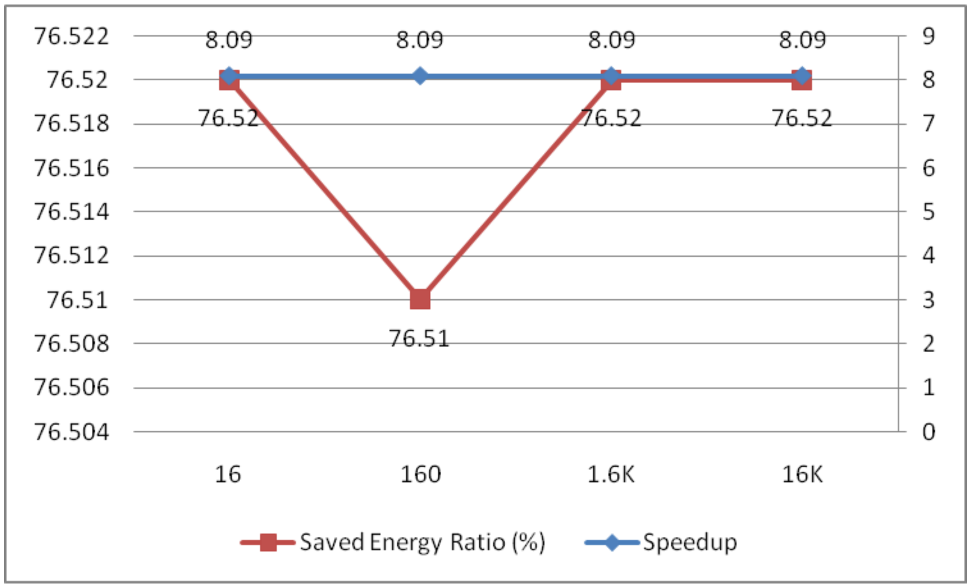


Figure V-12 Co-processor Design Data for Hadamard_3 Accelerator

Table V-10 Hadamard_4 FSL Performance and Energy Consumption Comparison

Input Size	Software Cycle	Hardware Cycle	Speedup	Software Energy Consumption (Joules)	Hardware Energy Consumption (Joules)	Saved Energy Ratio (%)
16	372	47	7.915	7.232	1.739	75.95
160	3720	470	7.915	72.317	17.381	75.97
1.6 K	37199	4700	7.915	723.165	173.806	75.97
16 K	371999	47000	7.915	7231.673	1738.052	75.97

There is only one register layer in this hardware IP design. Thus, it only has one clock cycle delay for the first data set. After that, results can be generated every clock cycle, and that is why there is no difference between different data sets.

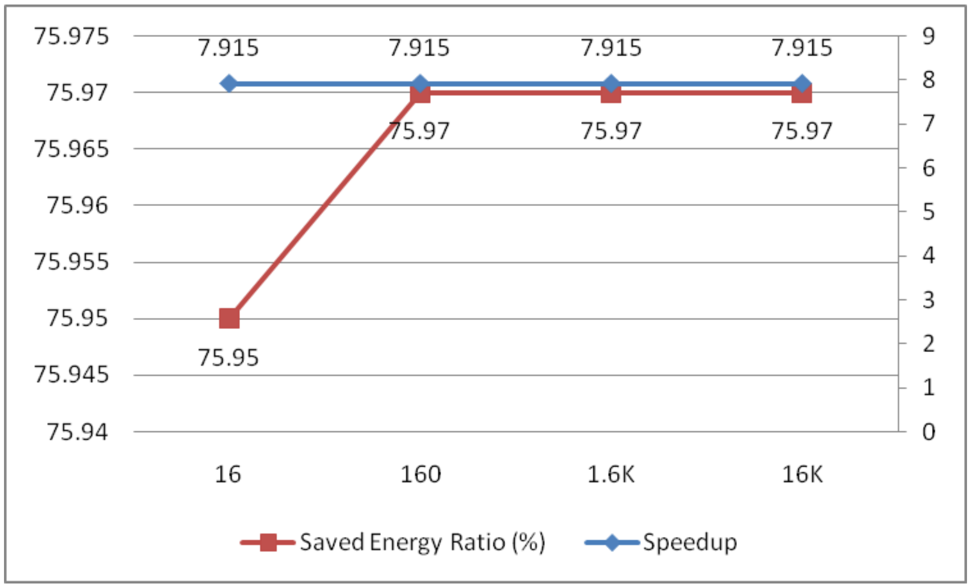


Figure V-13 Co-processor Design Data for Hadamard_4 Accelerator

Figure V-13 is the speedup and saved energy ratio for Hadamard_4. The average speedup for the Hadamard_4 is 7.915X, and the average saved energy consumption is 75.97%. Table V-11 is the system performance improvement and energy consumption comparison of the Hadamard_5.

Table V-11 Hadamard_5 FSL Performance and Energy Consumption Comparison

Input Size	Software Cycle	Hardware Cycle	Speedup	Software Energy Consumption (μJoules)	Hardware Energy Consumption (μJoules)	Saved Energy Ratio (%)
16	372	50	7.44	7.232	1.855	74.35
160	3720	473	7.865	72.317	17.521	75.77
1.6 K	37199	4703	7.910	723.165	174.217	75.91
16 K	371999	47003	7.914	7231.673	1740.992	75.93

Figure V-14 is the speedup and saved energy ratio for the Hadamard_5. With the help of registers for storing results, the Hadamard_5 also has longer execution time for the first data set. After that, results are generated every clock cycle. The difference between the Hadamard_2 and the Hadamard_5 is the delay for the first data set and resource costs. The Hadamard_2 costs even longer time to finish processing the first set of input data. The average speedup for the Hadamard_5 is 7.782X. The longer the system runs, the more energy can be saved by using the hardware IP. The average saved energy consumption is 75.49%.

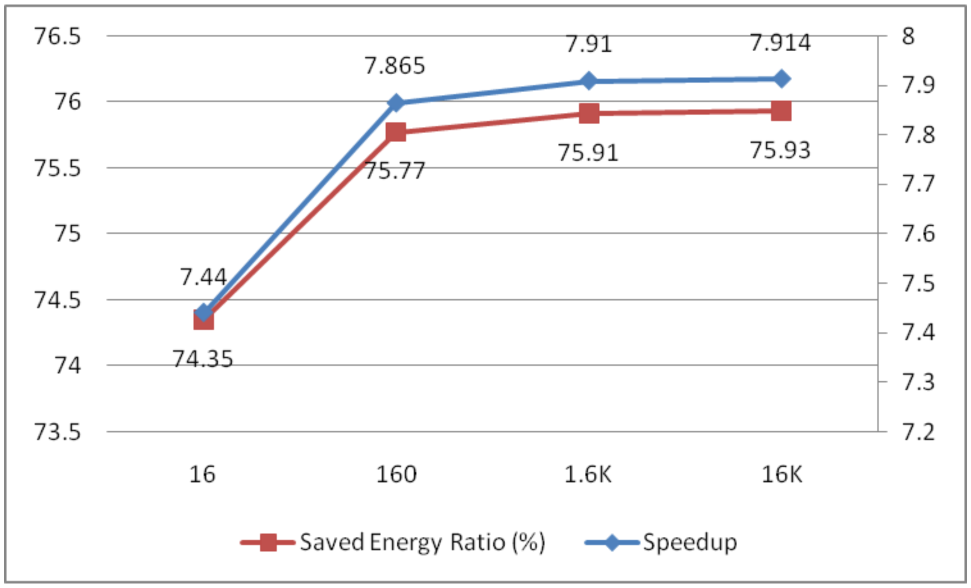


Figure V-14 Co-processor Design Data for Hadamard_5 Accelerator

5.6 Experimental Analysis

Five types of Hadamard IPs are designed and mapped into our hardware experimental platform. The resource costs, energy consumption and performance improvement of different designs are measured based on the Bus-IP design and Co-processor design.

Figure V-15 shows design costs of different Hadamard IPs [121]. Data can directly be read from figure V-15 that the Hadamard_2 costs the most on-chip resources, then following the Hadamard_1 IP. Because the Hadamard_2 uses a number of registers every processing step, it costs the most D Flip-Flops and other resources. The Hadamard_1 directly connects signals together to outputs, so it doesn't need D Flip-Flops. The Hadamard_3 is the fast Hadamard transformation, and it also directly connects signals to outputs, so the Hadamard_3 costs zeroes D Flip-Flop. The difference between the

Hadamard_4 and the Hadamard_5 is that for every processing layer of fast Hadamard transformation, the Hadamard_5 uses registers to store the value, while the Hadamard_4 only stores one layer value, signals of other layers also directly use wires for connections. Hadamard_3, Hadamard_4, and Hadamard_5 cost nearly the same amount of multiplexers and XOR gates, while Hadamard_1 and Hadamard_2 cost more resources.

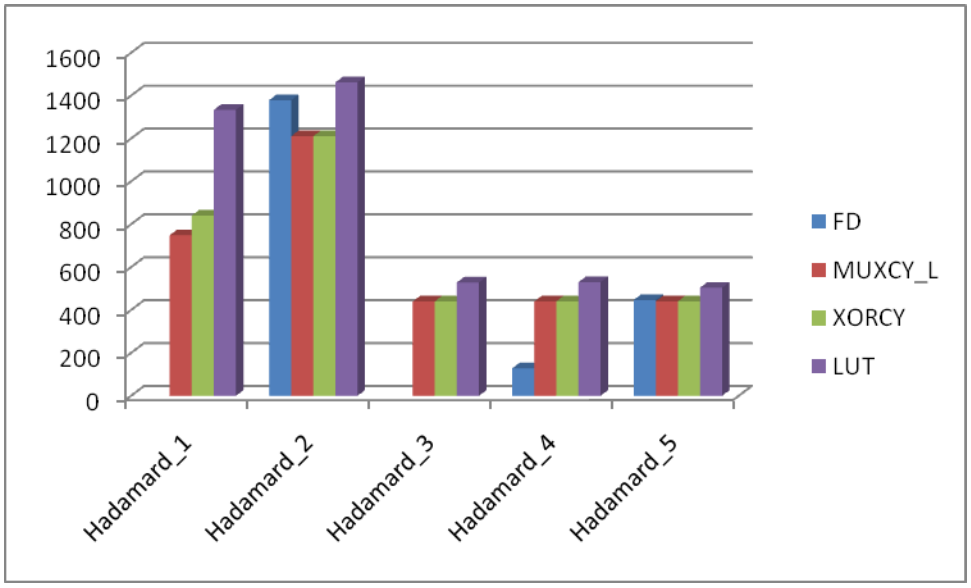


Figure V-15 Design Cost of Different Hadamard IPs

Figure V-16 is the comparison of system speedup based on the Bus-IP design and the Co-processor design. Both of these two designs improve the system performance. However, benefited from the high speed link of the MicroBlaze, the co-processor design achieves a higher performance.

For the Bus-IP design, all five types of hardware accelerators can reach a speedup around 2.8X. Among these designs, the speedup of the Hadamard_2 is the lowest; this is because the Hadamard_2 uses more hardware resources to build the hardware accelerator. It uses registers every step when finishing data calculation. Hadamard_1 and

Hadamard_3 have the same speedup when compared to the pure software design, because both of these two designs use wires to directly connect signals together to generate output results. In the hardware design, when designers use wires for connections, results are generated immediately. The Hadamard_5 is a three-layer pipeline design. It uses registers to temporarily store intermediate results. The Hadamard_5 costs longer time to finish the algorithm for the first data set, then, it generates results every clock cycle. Even though the Hadamard_4 is a simplified pipeline, it only uses one layer register for data storage. Other signals are connected by wires, so the Hadamard_4 costs shorter time to finish the algorithm than the Hadamard_5.

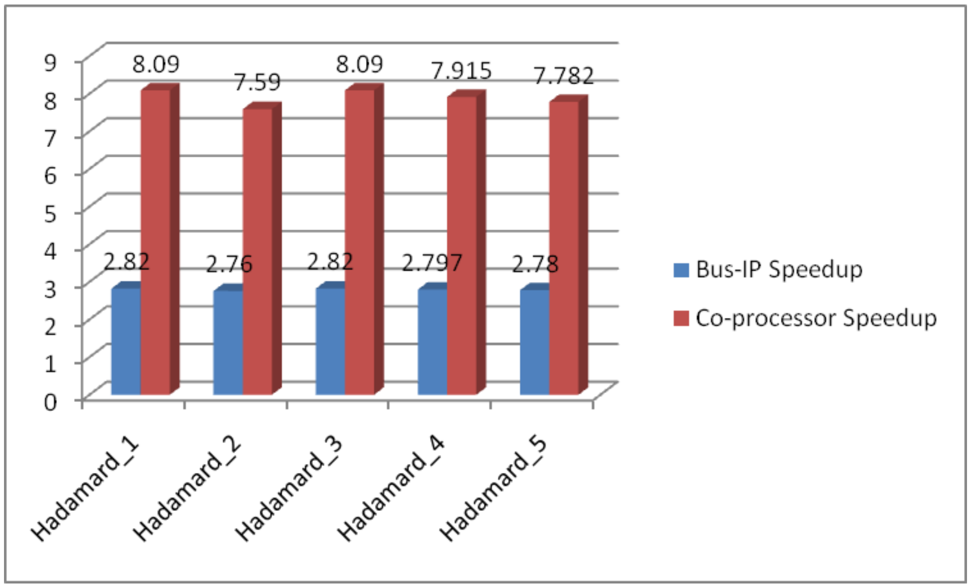


Figure V-16 Speedup Comparison of Bus-IP and Co-processor Design

The same phenomenon happens on the Co-processor design, and the average speedup can reach to 7.9X. The Hadamard_2 costs longest time. Following Hadamard_4 and Hadamard_5, Hadamard_1 and Hadamard_3 use the same time to finish processing data.

Figure V-17 is the saved energy of the Bus-IP and Co-processor design. The XPower analyzer is used to measure the energy consumption. The hardware energy consumptions are compared to the pure software method. From the figure, it is clearly that the Co-processor design saves more energy than the Bus-IP design.

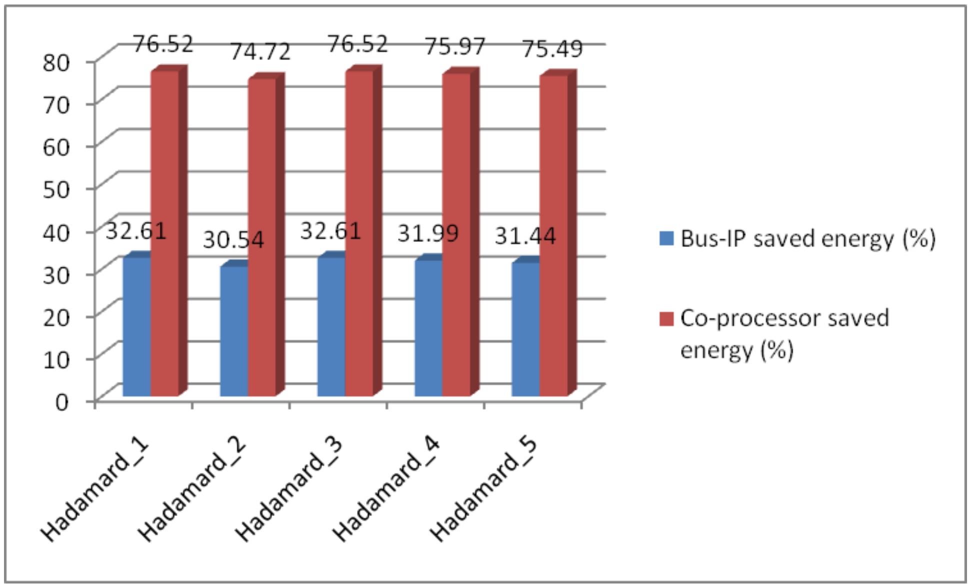


Figure V-17 Saved Energy of Bus-IP and Co-processor Design

Of the five types of hardware accelerators, the Hadamard_2 uses the most on-chip resources. It is obviously that the Hadamard_2 costs most energy. Based on the XPower analyzer, Hadamard_1 and Hadamard_3 use little power when the algorithm is running on the system, so both of them cost the least energy. They can save 32.61% energy when compared to the pure software method. Because Hadamard_4 and Hadamard_5 use registers to store the intermediate data, they cost more energy than Hadamard_1 and Hadamard_3.

Hadamard_1 and Hadamard_3 have the same speedup compared to the pure software method. They also save the same amount of energy for this specific algorithm.

However, based on the experimental results, the Hadamard_1 costs more on-chip resources than the Hadamard_3, so the Hadamard_3 is a better choice for accelerating the H.264 software application.

Because the Co-processor design uses a high speed link to connect the MicroBlaze to Hadamard IPs, it has better performance improvement compared to the Bus-IP design. Besides, the Co-processor design saves more energy than the Bus-IP design.

Regarding the maximum frequency of Hadamard accelerators, the Hadamard_1 has the lowest maximum frequency since it connects multiple adders together directly. However, the MicroBlaze on our hardware platform can reach 125MHz, if the CPU frequency increases; the system using the Hadamard_1 accelerator is not able to get correct results. Based on all the information collected from performance, energy consumption and resource costs, Hadamard_3 is the most suitable accelerator for our current SoC system.

5.7 **Summary**

In this chapter, after the identification of the hotspot function, which is the Hadamard algorithm, the hardware acceleration method is used to accelerate the chosen function.

Five typical types of Hadamard accelerators are designed. Each of them has their own features. Then all of them are mapped to our hardware experimental platform. The platform uses the soft-processor, called MicroBlaze, as the CPU. After providing the driver of the corresponding Hadamard IP, internal data can reach the hardware IP and been processed there.

Two types of hardware acceleration architectures are used. One is the Bus-IP design, the other is the Co-processor design. All Hadamard accelerators are mapped to these designs. System performance and energy consumption are collected. The system can reach 2.8X performance improvements and save 31.84% energy consumption by applying the Bus-IP design. The Co-processor design can have 7.9X performance and save 75.85% energy consumption. The optimum Hadamard accelerator, which is the Hadamard_3 for accelerating the H.264 software application, can be found.

VI CHIP DESIGN AND SYSTEM VERIFICATION

In order to make a real Integrated Circuit (IC), designers analyze the software application at first. They use software profiling tools to generate the statistics of a software application. Then, the software profiling method is used to identify the hotspot function, which is the bottleneck of the aimed software application. After that, the hotspot function is transformed to the corresponding hardware accelerator. In this research, hardware accelerator is tested under the FPGA based SoC system. The proposed hardware acceleration method can efficiently create a balanced design and reduce the design cycle. If the designed hardware accelerator can achieve a higher performance and keep the energy consumption and resource costs at an acceptable level. This hardware accelerator can be viewed as the candidate to be made into a real IC.

6.1 IC Design Flow

There are several steps of manufacturing an IC chip. Figure VI-1 is the basic flow of the IC design. Although there may be other kinds of design flows, the basic rules are the same.

Designers put different efforts in the process of making a successful IC. The most important stage in the design flow is the information acquisition stage. Designers get the idea of the design. They do some research on the aimed design, including specifications, algorithms, and even the architecture. With a clear understanding of the design, designers can go ahead for the next stage. Good designers must have sufficient knowledge in their specific areas and keep on accumulating them.

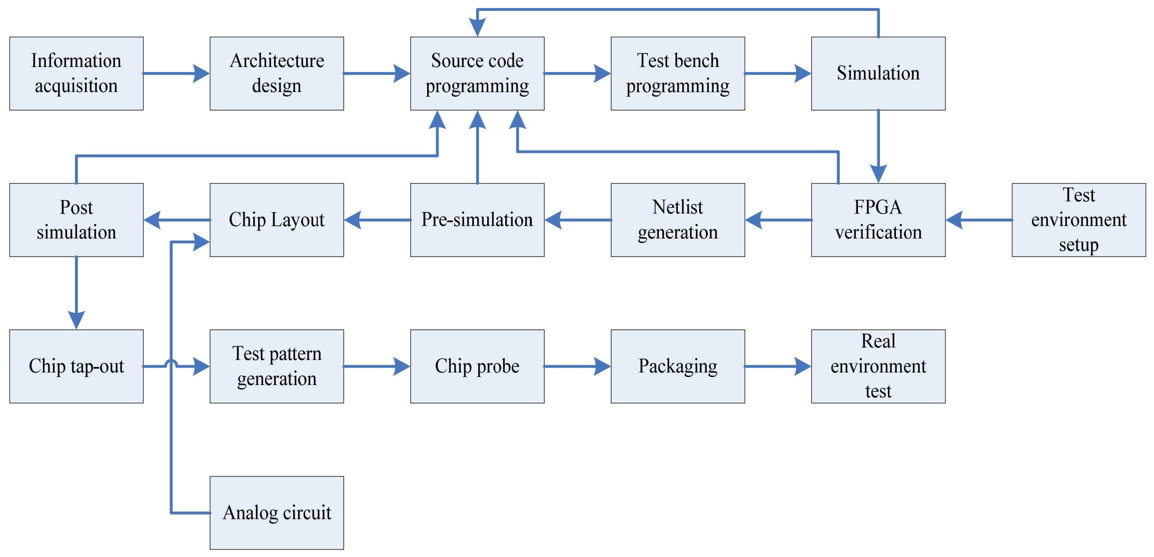


Figure VI-1 IC Design Flow Chart

In the architectural design stage, designers must be familiar with all the related knowledge of the design. The selected algorithm is directly related to the structure, this is why we need to first verify and generate the most suitable algorithm form. The best architecture is the one with the fastest speed, the lowest power consumption, and the minimum chip size. Designers can also use some IPs, which are not designed by themselves. Some IPs cannot be used directly, thus, designers must modify them to fit their requirements.

Hardware designers use HDLs to write the source code. In HDLs, every statement is one circuit. All the statements work at the same time no matter their positions. Thus, designers must be very careful to avoid mistakes. Designers must pay attention to their code and elaborate their coding abilities. They must also consider the future tests of the design, the reusability, the manufacturability and low power consumption.

After that, designers must write test benches for their designs. Test bench provides simulation models for the design. For the large ASIC design, coding the test bench may take much longer time than coding the ASIC itself. Usually, the test bench should cover as more cases as possible to make sure the design works successfully.

Then it goes to the Register Transfer Level (RTL) simulation, which is also called behavioral simulation. It is based on the RTL function but no timing considerations. All designs must pass RTL simulations, and then the source code can be processed for the following stages. The goal of the simulation is to find out errors before the hardware manufacturing. That is why it may take many rounds to modify the code and re-do the simulation. If a designer uses an FPGA to develop the circuit, the design code can be synthesized into the FPGA netlist.

The FPGA is nearly the same as the real IC except timing constraints and IPs. The FPGA verification with the real test environment can find most of the problems. The FPGA verification usually takes the longest time in the entire design flow. Nearly all the real problems can be found in the verification stage. The source code cannot be tapped-out without a successful verification.

The design code is viewed as a good one once it passes the FPGA verification. Then the code is synthesized into a netlist for the chip layout. After that, designers do simulation again, which is called pre-simulation. The target of the simulation is not source code anymore, but the netlist. On pre-simulation stage, timing issues are added to the simulation, so the simulation is closer to the performance of the real chip. The timing is only added to the cells and registers, but not wires. If pre-simulation results are not as expected, then the timing cannot be met, and the source code should be re-designed. The netlist is ready for layout if the design passes the pre-simulation. With the analog circuit, designers can finish the layout writing.

After the chip layout process, another netlist is generated according to the real wires of the chip. With the consideration of the wire delay, the design goes to the post-simulation stage. In most cases, post-simulation results are nearly the same as the pre-simulation results. However, if results of the post-simulation does not meet system requirements, designers must go back to modify the source code and re-do the flow again. If post-simulation results meet design requirements, the design is ready for the chip fabrication. Usually it will cost huge money for the process from tape-out to chip.

The manufactured chip is packaged and mounted to check whether it is good or not. The test pattern should have higher coverage for all the possibilities. The higher the coverage, the better of the yield can be achieved. The tester uses the probe card to check the chip on the wafer. When the chip passes the chip probe stage, the wafer can be cut down for packaging. After packaging, IC chips are tested again to make sure they are good. The last step in the entire flow is to test the IC mounted on the real system.

6.2 Design for Edge Detection

6.2.1 Edge Detection for Remote Control

The technology evolves rapidly. Currently, people enjoy many high-tech products such as remote control through digital cameras. It is known that the edge detection plays an important role in the remote control process. The edge detection part processes the input data and extracts the key feature for further steps. There are several edge detection methods. In this research, the Sobel edge detection is designed and verified by using the verification platform [113].

The edge detection is the most commonly used approach to detect discontinuities in gray level by far. An edge is a number of pixels, which lie on the boundary between two regions. Edges are located in the areas with strong intensity contrasts. One of the most popular ways to perform edge detection is by the gradient. There are many edge detection operators based on the gradient detection, and the Sobel operator is chosen to be used and a verification platform is used to test the designed hardware.

The Sobel operator is a 3×3 mask used to compute the gradient of the corresponding region. Figure VI-2 shows the pixels of the aimed image region, the Sobel operator multiplies with image pixels to find out the gradient at the point labeled p5.

p1	p2	p3
p4	p5	p6
p7	p8	p9

Figure VI-2 Image Region for Edge Detection

The edge detecting computations based on Sobel operators are as follows:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \times A \qquad \text{VI (1)}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \times A \qquad \text{VI (2)}$$

As shown in the equations VI (1) and VI (2), A is the grayscale of the original images, G_x is the row gradient and G_y is the column gradient. An approach used frequently is to approximate the magnitude of the Sobel gradient by absolute values:

$$\nabla f \approx |G_x| + |G_y| \qquad \text{VI (3)}$$

After generating the value from the equation VI (3), the result is compared with a threshold, which sets the value to either black or white. The result is sent back to the image point labeled p5.

6.2.2 Sobel Operator Design

There are several previous researches about the edge detection design using the Sobel operator. In [114], S. Halder *et al.* designed a Sobel operator based on the optimized algorithm. However, their design used too many dividers and multipliers. Thus, too many on-chip resources are used without significant performance improvement. Besides, they did not have a complete solution for the Sobel edge detection. T. A. Abbasi *et al.* proposed an FPGA-based architecture for the Sobel edge detection operator [115]. However, they used two RAM to store the data, one for the original data, and the other for the result data. Thus, the resource consumptions are too high. [116] proposed that they could operate the Sobel operation efficiently. However, their design can only run at a very low frequency, and they still need to use two storage spaces to save the data. Another design based on the Sobel operator was described in

[117]; the author designed a massive pipeline for algorithm calculations. This design can increase the performance significantly. However, the SoC may not able to allocate enough resources for this design. It is not a balanced design scheme. V. Sanduja and R. Patial designed a complete edge detection system based on the Sobel operator, and their design had an accurate result for each pixel. However, the costs of the resources are still a little higher by using two separate memory parts to store the data [118].

In this research, we use the Sobel operator to detect the edge of a 256×256 grayscale image. The Sobel operation is separated into two parts, one part is a Sobel core, and the other part is the Sobel full scan. The Sobel core is a single calculation of the matrix, and the Sobel full scan is used to scan the full image frame [120]. The Sobel operator design is optimized for the hardware implementation of following aspects:

- The Sobel core needs two clock cycles to finish the calculation in this design. In order to make the design run efficiently, the operation defines that one pixel data is loaded to the Sobel core every clock cycle. After loading the input data, the Sobel operation can generate the output data in the following cycle. The Sobel core of this design needs two clock cycles to generate the result, so the Sobel core part is connected to a clock, whose frequency is two times as the clock connected to the Sobel full scan. Thus, the data can flow continuously without latency.
- The other optimization is to put the generated data to the image pixel labeled p1 instead of p5. The design can have a higher efficiency by using this way. After the calculation of the single matrix. The Sobel mask will move to the

next window, it will go through all rows and columns in the image frame. If generated results are sent back to p5, the Sobel mask must extract the original pixel before results are sent back. If designers want to avoid the influence of results, they must use other storage devices to hold results, which is a costly method. However, we store results directly in the pixel labeled p1. The pixel labeled p1 will not be used in the future, so it has no influence for the future calculation even if the data is modified. Moreover, the designer does not need additional storage devices for calculation results.

6.2.3 Experimental Results

Our Sobel edge detection design is divided into two pieces: one is the single matrix calculation; the other is the full frame calculation. Because the single calculation needs time to process the data, the clock frequency of the internal single calculation is twice as the full frame calculation to make the whole data run as a pipeline. In the real case, each gate has its own timing constraint. One can only use around the maximum of six adders or subtractors together to generate the output data through combinational logics. Thus, the single matrix calculation is separated into two RTL blocks. This can not only have the least register usage, but also a relatively higher frequency of the designed circuit.

Compared to the single Sobel operator design in [114], they designed a single Sobel operator and mapped the design on the Xilinx Spartan 3 XC3S50-5PQ208 board. Their design can reach up to 190MHz frequency. In contrast, our design can only reach up to 156MHz on the same board. However, as for the resource costs, our design only occupies 10% of on-chip slices, while their design cost up to 16% of on-chip slices. In the SoC system, on-chip resource costs are key factors, which have great impacts on the design. The less of the resource cost, the lower of the power consumption can be reached. If a 256 × 256 frame need to be processed, our design can ideally consume 0.41ms to finish, while their design can ideally consume 0.34 ms to finish. This is still an acceptable time latency in our remote control system; especially consider the saved on-chip resources.

Table VI-1 Device Utilization Comparison

	Number of Occupied Slice	Number of Slice Flip-Flops	Number of 4-input LUTs
Design in [112]	1987	836	3901
Proposed Design	1144	128	1400

In another design from V. Sanduja and R. Patial [118], a 20 × 40 picture was processed by the Sobel edge detection design. Their design used Xilinx Virtex 4 FPGA board. The device was XC4VLX200, and the package was FF1513. Even they got the accurate result for each pixel, that design costs too many on-chip resources. Table VI-1 shows the device utilization comparison between our design and their design in [118]. It is shown that our design uses much less resources than their design. One advantage of our design is using a single RAM to store the data, after pixels are processed by the single matrix calculation. The processed result is sent back to the position labeled p1 instead of p5. This optimization method can save a large amount of storage space and the processed pictures are usable for further steps in our SoC design. The other advantage is that our design does not process the rightmost two columns and lowest two rows of the picture. This method can save the processing time when the data set is huge enough. Besides, the omitted pixels have little influence to final results.

It is proved that by using optimization methods mentioned above, the design can reach a higher performance by applying the pipeline technique. At the same time the resource costs are kept at an acceptable level. Figure VI-3 shows simulation results of the designed Sobel edge detection. It proves that the pixel data can be fetched every clock cycle and then results are written back in the following clock cycle.

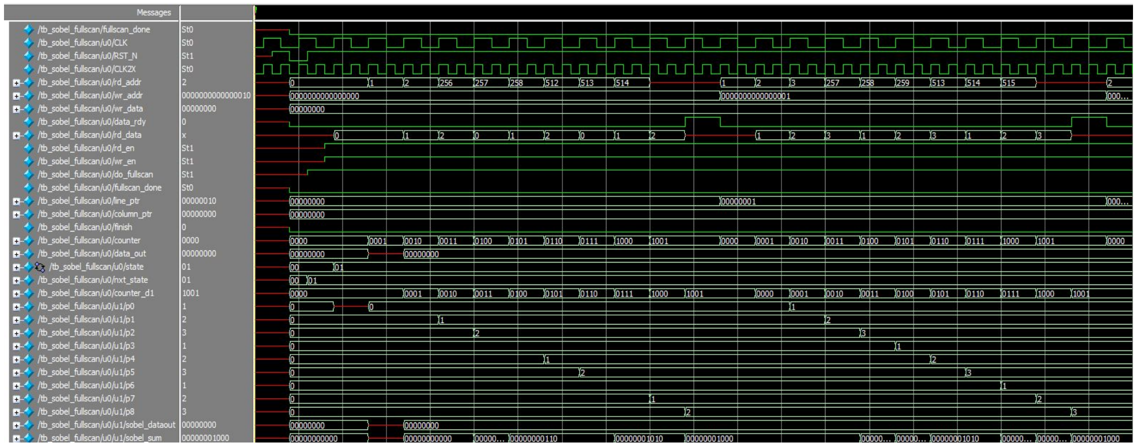

Messages
/tb_sobel_fullscan/fullscan_done St0
/tb_sobel_fullscan/u0/CLK St0
/tb_sobel_fullscan/u0/RST_N St1
/tb_sobel_fullscan/u0/CLK2X St0
/tb_sobel_fullscan/u0/rd_addr 2
/tb_sobel_fullscan/u0/wr_addr 0000000000000010
/tb_sobel_fullscan/u0/wr_data 00000000
/tb_sobel_fullscan/u0/data_rdy 0
/tb_sobel_fullscan/u0/rd_data x
/tb_sobel_fullscan/u0/rd_en St1
/tb_sobel_fullscan/u0/wr_en St1
/tb_sobel_fullscan/u0/do_fullscan St1
/tb_sobel_fullscan/u0/fullscan_done St0
/tb_sobel_fullscan/u0/line_ptr 00000010
/tb_sobel_fullscan/u0/column_ptr 00000000
/tb_sobel_fullscan/u0/finish 0
/tb_sobel_fullscan/u0/counter 0000
/tb_sobel_fullscan/u0/data_out 00000000
/tb_sobel_fullscan/u0/state 01
/tb_sobel_fullscan/u0/nxt_state 01
/tb_sobel_fullscan/u0/counter_d1 1001
/tb_sobel_fullscan/u0/u1/p0 1
/tb_sobel_fullscan/u0/u1/p1 2
/tb_sobel_fullscan/u0/u1/p2 3
/tb_sobel_fullscan/u0/u1/p3 1
/tb_sobel_fullscan/u0/u1/p4 2
/tb_sobel_fullscan/u0/u1/p5 3
/tb_sobel_fullscan/u0/u1/p6 1
/tb_sobel_fullscan/u0/u1/p7 2
/tb_sobel_fullscan/u0/u1/p8 3
/tb_sobel_fullscan/u0/u1/sobel_dataout 00000000
/tb_sobel_fullscan/u0/u1/sobel_sum 0000001000

Figure VI-3 Fast Hadamard Transformation of Eight Vectors

6.3 System Verification Platform Design

The designed hardware must be tested under the test environment. In this research project, a system verification design platform is created. The platform can be used to test the functionality of the design. The platform includes the following components:

- Test module design. The Design Under Test (DUT) module which is the edge detection module designed before. It will be connected to the platform as the slave module.

- Verification module design. The functional simulation of the behavior of the CPU is used to test the designed hardware. The memory controller and some external interface modules are designed and verified.

- Advanced Microcontroller Bus Architecture (AMBA) based protocol design. Advanced High-performance Bus (AHB) from the AMBA bus protocol is used for data transmission [119].

- Functional register configuration design. A designed hardware must have the configuration registers for the software designer to design the corresponding software application.

Figure VI-4 is the system verification platform. The verification platform includes the CPU, the memory controller, the DUT, and other interfaces. The platform uses AHB as the communication bus. Each DUT must have the slave interface and the master interface for communication through the AHB bus. As the central processing unit of the system, the CPU has only the master interface to send commands. As the data storage devices, memories are viewed as slaves. Thus, they only have slave interfaces.

The master interface is used to send commands to slave interfaces and receive the data or responses from slave interfaces. In the system design, the CPU will initial the command to the DUT, which is the Sobel edge detection module in this project, through the DUT's slave interface. The command is used to configure the functional registers of the designed hardware IP. Once the designed hardware gets the command from the CPU, it will extract the information in the command and take further actions. The information includes the start and the stop command of the DUT, the initial address of the memory, or other peripherals, where the DUT can fetch data from there, etc. Then, the DUT can fetch data from memory or other peripherals through its master interface to the slave interface of the memory controller. After finishing processing, the DUT can send results back to the memory or other peripherals if necessary.

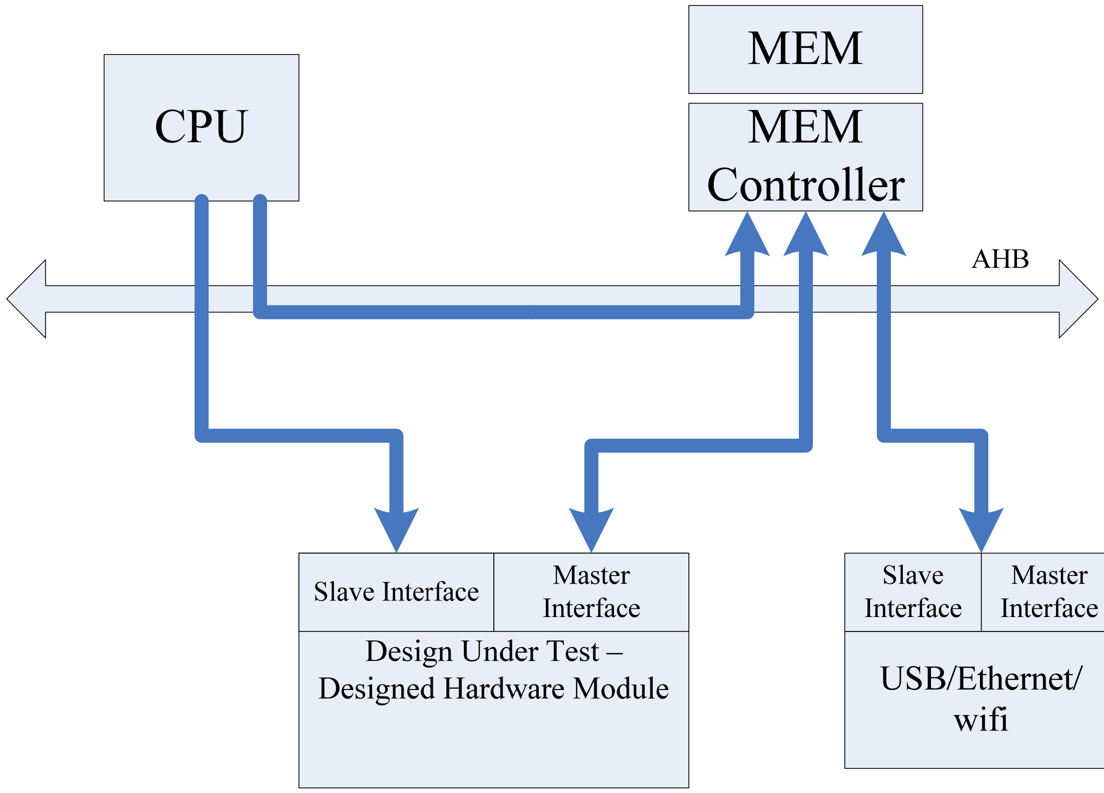


Figure VI-4 System Verification Platform

In this research project, the CPU is instantiated as a functional module, and it sends command registers to the Sobel edge detection operator. The image frame used for the experiment is a 256x256 grayscale picture, so there should be 65536 positions in the memory. The Sobel operator extracts the information from command registers, so that the operator knows when to write and read data. The Sobel edge detection operator also gets the information of where to fetch data and the amount of data. Then, the Sobel operator sends commands to the memory controller to fetch the data and do calculations. The system will send results back to the memory after finishing the whole image frame calculations.

The AHB bus protocol in this project is an industry standard, so that the platform can be used for other DUTs in the future if properly configured. The transmission on the AHB bus is 32bit. The responses of slaves are set to okay for easy use to ensure the communication is good for this design.

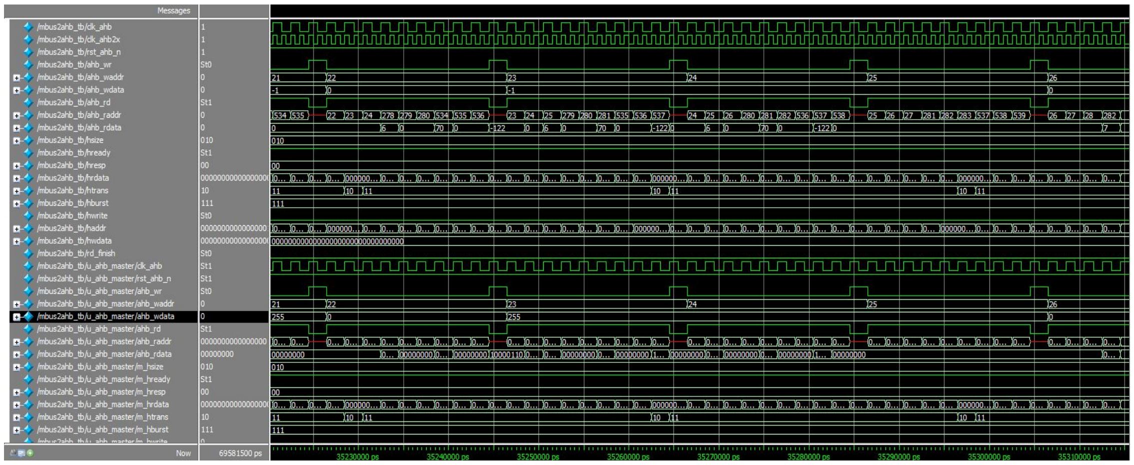

Messages
/mbus2ahb_tb/clk_ahb 1
/mbus2ahb_tb/clk_ahb2x 1
/mbus2ahb_tb/rst_ahb_n 1
/mbus2ahb_tb/ahb_wr St0
/mbus2ahb_tb/ahb_waddr 0
/mbus2ahb_tb/ahb_wdata 0
/mbus2ahb_tb/ahb_rd St1
/mbus2ahb_tb/ahb_raddr 0
/mbus2ahb_tb/ahb_rdata 0
/mbus2ahb_tb/hsize 010
/mbus2ahb_tb/hready St1
/mbus2ahb_tb/hresp 00
/mbus2ahb_tb/hrdata 0000000000000000
/mbus2ahb_tb/htrans 10
/mbus2ahb_tb/hburst 111
/mbus2ahb_tb/hwrite St0
/mbus2ahb_tb/haddr 0000000000000000
/mbus2ahb_tb/hwdata 0000000000000000
/mbus2ahb_tb/rd_finish St0
/mbus2ahb_tb/u_ahb_master/clk_ahb St1
/mbus2ahb_tb/u_ahb_master/rst_ahb_n St1
/mbus2ahb_tb/u_ahb_master/ahb_wr St0
/mbus2ahb_tb/u_ahb_master/ahb_waddr 0
/mbus2ahb_tb/u_ahb_master/ahb_wdata 0
/mbus2ahb_tb/u_ahb_master/ahb_rd St1
/mbus2ahb_tb/u_ahb_master/ahb_raddr 0000000000000000
/mbus2ahb_tb/u_ahb_master/ahb_rdata 00000000
/mbus2ahb_tb/u_ahb_master/m_hsize 010
/mbus2ahb_tb/u_ahb_master/m_hready St1
/mbus2ahb_tb/u_ahb_master/m_hresp 00
/mbus2ahb_tb/u_ahb_master/m_hrdata 0000000000000000
/mbus2ahb_tb/u_ahb_master/m_htrans 10
/mbus2ahb_tb/u_ahb_master/m_hburst 111
Now 69581500 ps
35230000 ps
35240000 ps
35250000 ps
35260000 ps
35270000 ps
35280000 ps
35290000 ps
35300000 ps
35310000 ps

Figure VI-5 Sobel Operator with AHB Bus

Figure VI-5 is simulation results of the Sobel operator with the AHB bus. It proves that the design works as expected in the system. The data is loaded from external memory to the local cache in order to increase the system performance. With the verification platform, the Sobel operator is proved to be a good one and is ready to go ahead for the following stages in the IC design flow.

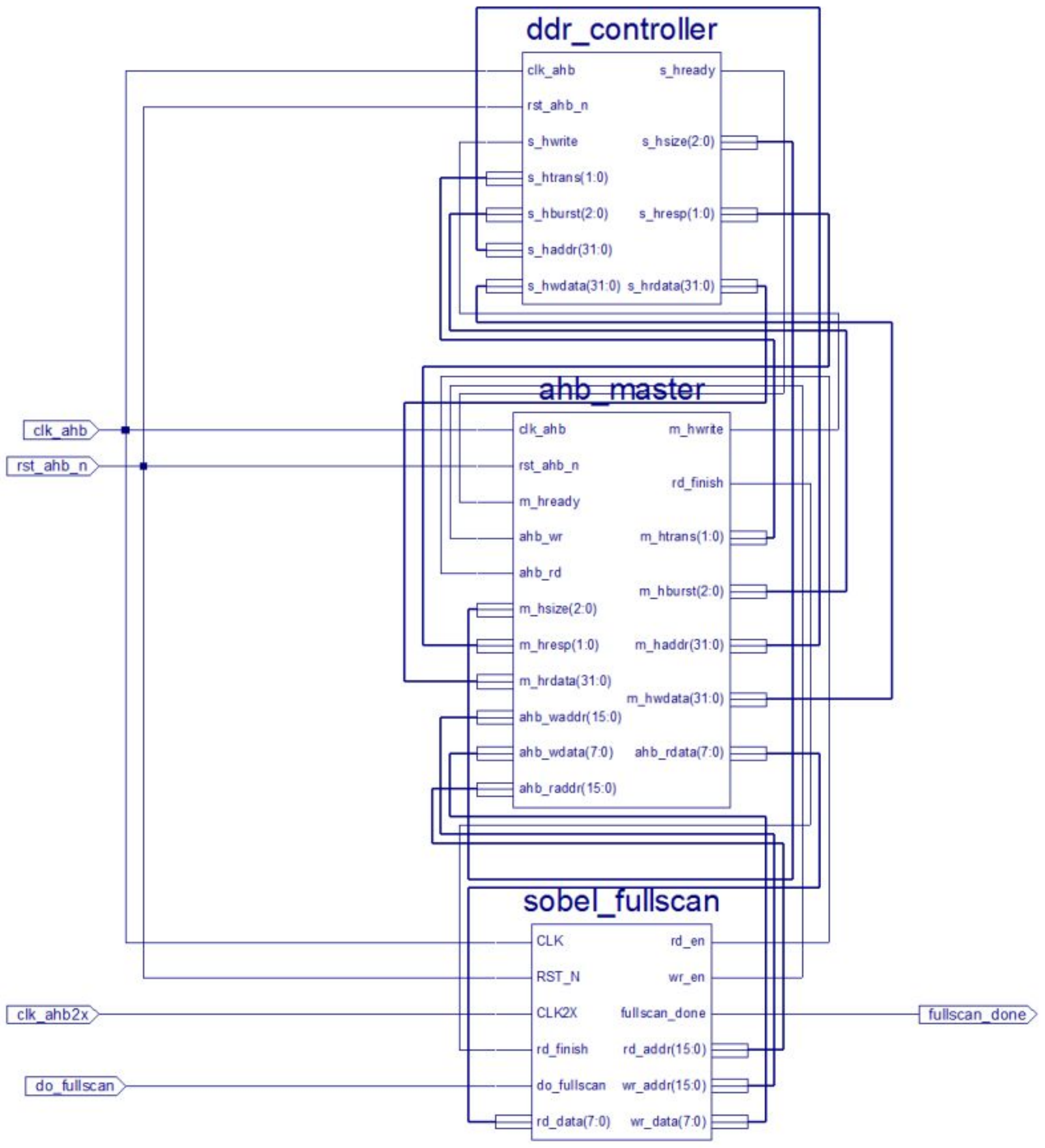


Figure VI-6 Schematic of Sobel Operator with AHB Bus

Figure VI-6 is the block design of the system. The system uses two synchronized clocks and one global reset signal. Two signals, which are do_fullscan and fullscan_done, are reserved for future usage. If there are multiple DUTs in the system, an arbiter will be used to accommodate different DUTs based on AHB protocol. The Sobel operator communicates with the off-chip memory through the ddr_controller. The Sobel core IP is integrated in the sobel_fullscan IP.

6.4 Summary

In this chapter, the system verification platform is introduced. The Sobel edge detection operator is designed and verified on the verification platform.

After identifying the hotspot function of the software application, designers can proceed to make the IC chip. The IC design flow is introduced to make sure the designed IC chip is a good one. The Sobel edge detection operator is selected because it is one of the most commonly used methods in the video remote controller design. The Sobel operator is separated into two parts in order to process data more efficiently and cost less resource.

The system verification platform is set up to verify the designed hardware. The verification platform is composed of the functional CPU module, the DUT module, the memory module, and other peripherals. These modules communicate with each other through the AHB bus protocol. It is proved that the designed Sobel operator works efficiently on this verification platform. The designed Sobel operator can be processed to the next stage to make an IC chip. The verification platform can be further used to verify other designed hardware designs.

VII CONCLUSION & FUTURE WORK

7.1 Concluding Remarks

This dissertation described a hardware acceleration method used for accelerating the software application on the SoC system, and the design of the system verification platform.

Conventional hardware acceleration methods are very specific to the aiming software. It works well if the project is not very large. However, for large design projects, there is a demand for a theory guide to accelerate the design.

1) Designers usually directly accelerate the software based on the execution time of functions. However, this approach does not always lead to optimum performance. In our proposed software profiling methods, designers must not only focus on the execution time of each function, but also the number of loops of a specific function and cycle consumptions per loop.

   It has been stated that even though some functions cost more execution time, it may not be suitable to be accelerated. Some functions will have small performance improvement when implemented as hardware IPs. Even though they execute a lot of loop rounds and have higher execution time, they are not good candidates to be accelerated. Some functions may cost a lot of execution time, and they cost lots of cycles per loop. However, with few loop rounds, it may be a waste of hardware resources. Thus, they are not suitable to be accelerated also. A function that is suitable to be accelerated is the one that has a large number of cycles per loop and loop rounds. With longer execution time, this function could have higher a priority to be accelerated.

2) For the hardware acceleration method, there are three types of hardware acceleration methods that can be used to implement the hotspot function. Each hardware acceleration method has its own benefits and limitations. Bus-IP design and Co-processor designs are the two mostly used acceleration methods.

   The hotspot function is implemented to hardware using the hardware description language. Then, it is wrapped as a hardware IP. After that, the hardware IP is mapped to an FPGA platform. Every hardware IP is assigned to a dedicated memory address range within the system. When the CPU wants to communicate with the hardware IP, it must find the exact memory address for the specific hardware IP.

   The Driver for the hardware IP is provided in the software. The data need to be processed is sent to the hardware IP for processing. After finishing the processing, results are sent back to the CPU.

3) In order to design a balanced, optimized SoC system, designers must consider not only the system performance gain, but also the energy consumption and resource costs. Different SoC systems have different demands. For example: When building a mobile SoC system, it is important to take into account the system power consumption as the first priority. If we want to build a general

SoC system that can be used in most cases, speed is a key factor. There is always a trade-off among system performance, energy consumption, and resource costs. Designers must balance these factors for an efficient SoC system. If the designed system cannot meet system requirements, designers must re-check and balance the whole system. Sometimes they must re-design the hardware IP and evaluate it until the designed IP can fully meet system requirements.

H.264 video CODEC is used as the software application that is accelerated. First, software methods are used to profile the H.264 software application. All the statistics of H.264 internal functions are collected. The HadamardSAD has been found as the hotspot function. Then, the Software function is implemented into a hardware IP using Verilog hardware description language. Five types of hardware IPs are designed based on the original Hadamard transformation and fast Hadamard transformation for the aimed hardware architecture. After that, each Hadamard IP is successfully mapped to the FPGA platform using the Bus-IP design and Co-processor design.

Different system attributions for each design are measured. Simulation results are generated to compare the execution time between both software and hardware methods. The power analyzer is used for measuring the overall energy consumption. Resource costs are generated using the synthesis tool. Different results are compared from observing results based on performance, energy consumption and resource allocations. It is concluded that the wired fast Hadamard transformation is the optimum solution for our current hardware platform.

4) The system verification platform is designed based on the IC design workflow. After identifying the hotspot function and selecting the most suitable hardware accelerator. Designers need to proceed to make a real IC chip. The Sobel operator in the edge detection is chosen for further steps. Two optimization methods are used in the Sobel operator design. The hardware pipeline technique is used in order to make the operator process as much data as possible. Furthermore, the Sobel operator is modified to minimize the hardware resource costs.

   One platform, which uses industry AHB protocol for communication, is created for the system verification. The verification platform is composed of different blocks, including the functional CPU block, the DUT block, the memory block, and the memory controller block. The data and commands are sent through the AHB bus among different blocks.

7.2 Future Work

This dissertation discusses a complete hardware acceleration method workflow. It provides a guideline for designers who aim to promote the system performance. However, there are also many ways to improve it.

- Modifying the CPU instruction set. At present, we use the Bus-IP design and Co-processor design to accelerate the Hadamard function. There is another type of the hardware acceleration method, called the CPU instruction set modification. However, the CPU being used doesn’t allow modification. In the future, it is possible to find a CPU that can be modified, and then a full comparison of different hardware acceleration methods can be made and meaningful conclusions can be generated.

- Taking a step further, currently, nearly all the processes are handled manually step by step. We will try to make the process more automatic. System C is a good tool for describing functional behaviors. After coding the desired hardware accelerator based on the system C, an automatic tool could be used to directly transfer it to the corresponding hardware. This will make the design process more acceptable by software engineers. Thus, it will make the software-hardware co-design more attractive to designers.

- In the future design, the off-chip memory can be used as the data storage device. In our system, the MicroBlaze can fetch the data directly through XCLs, and it can also get the data through a system bus. Hardware acceleration behaviors using the off-chip memory can be analyzed. If the data transfer happened between hardware IP and off-chip memory, the Bus-IP design can have better performance than the Co-processor design. If the hardware IP in co-processor design requests data from the off-chip memory, it will impose great impacts on the system bus, and CPU behaviors.

- More hardware optimization techniques can be used in the verification platform. The system bus can be split to two separate buses. One is dedicated to the CPU with a higher frequency, so that the performance cannot be affected if the bus frequency is low. The other bus with a relatively lower frequency is dedicated for communication among the memory block and other peripherals. Also, more DUTs can be added to the verification platform, so that the communication among different blocks are important issues to be studied. Furthermore, the verified hardware design should be continued for other steps based on the IC design workflow.

REFERENCES

[1] The international technology roadmap for semiconductors: 2010 edition, 2010. http://www.itrs.net/Links/2010ITRS/Home2010.htm.

[2] G.E. Moore, “Cramming more components onto integrated circuits. Electronics,” 38(8), Apr. 1965.

[3] "Excerpts from A Conversation with Gordon Moore: Moore’s Law," (PDF). Intel Corporation. 2005. p. 1. Retrieved 2006-05-02.

[4] "Moore's Law Predicts the Future of Integrated Circuits," Computer History Museum. 2007. Retrieved 2009-03-19.

[5] Mead, C., Conway, L. “Introduction to VLSI systems,” Reading, Mass. Addison-Wesley Publishing Co., 1980. 426 p.

[6] Weik, Martin H. “A Third Survey of Domestic Electronic Digital Computing Systems,” Ballistic Research Laboratories. 1961.

[7] Introduction of System-on-Chip, [Online]. Available: http://en.wikipedia.org/wiki/System-on-a-chip.

[8] R. Saleh, S. Mirabbasi, G. Lemieux, et al., “System-on-Chip: Reuse and Integration,” in Proceedings of IEEE, vol. 94, Issue 6, pp. 1050 - 1069, June 2006.

[9] R. Rajsuman, “System-on-a-Chip Design and Test,” Boston, MA: Kluwer, 2000.

[10] H. Chang, L. Cooke, M. Hunt, et al., “Surviving the SoC Revolution: A Guide to Platform-Based Design,” Boston, MA: Kluwer, 1999.

[11] IanKuon, R. Tessier, J. Rose. “FPGA Architecture: Survey and Challenges,” Journal Foundations and Trends in Electronic Design Automation, Volume 2 Issue 2, 2008,2. Pp 135-253.

[12] Yovits, Marshall C. “Advances in computers,” 37. Academic Press. 1993. pp. 105–107.

[13] Liptak, Béla G. “Instrument Engineers' Handbook: Process control and optimization,” 2. CRC Press. 2006. pp. 11–12.

[14] IBM Microelectronics, Blue Logic SA-27E ASIC Product Brief, June 2002.

[15] IBM Microelectronics, Blue Logic Cu-08 ASIC Product Brief, April 2002.

[16] IBM Microelectronics, Blue Logic Cu-11 ASIC Product Brief, June 2002.

[17] Wiśniewski, Remigiusz. "Synthesis of compositional microprogram control units for programmable devices," Zielona Góra: University of Zielona Góra. 2009. pp. 153. ISBN 978-83-7481-293-1.

[18] G. Martin, H. Chang. "Winning the SoC revolution: experiences in real design," English, Boston: Kluwer Academic Publishers, 2003.

[19] M. Keating, P. Bricaud, "Reuse Methodology Manual: For System-on-a-Chip Designs," 3rd ed. Boston, MA: Kluwer, 2002.

[20] Computer History Museum [Online]. Available: http://www.computerhistory.org/semiconductor/timeline/1974-digital-watch-is-first-system-on-chip-integrated-circuit-52.html.

[21] Yen-Kuang Chen, S. Y. Kung. "Trend and Challenge on System-on-a-Chip Designs," Journal of Signal Processing Systems. vol. 53. Issue 1-2. Pp 217-229. Nov.2008.

[22] AMBA Specification (Rev 2.0), ARM Ltd., May 13, 1999.

[23] Open Core Protocol Specification 2.1. OCP-IP Assoc. [Online]. Available: http://www.ocpip.org.

[24] Virtual Socket Interface Alliance. [Online]. Available: http://www.vsi.org.

[25] M. Richhetti, "Overview of proposed IEEE P1500 scalable architecture for testing embedded cores," (slide presentation) presented at the Design Automation Conf., Las Vegas, NV, 2001, pp. 1–26.

[26] Magarshack P. "System-on-chip beyond the nanometer wall," Proceedings of Design Automation Conference. Pp 419-424. June, 2003.

[27] B. Brock and K. Rajamani, "Dynamic Power Management for Embedded Systems," in Proc. of Int'l SOC Conf., pp. 416 - 419, Sept. 2003.

[28] J. Choi and H. Cha, "Memory-Aware Dynamic Voltage Scaling for Multimedia Applications," IEE Proceedings of Computers and Digital Techniques, vol. 153, no. 2, 2006, pp 130-136, Mar.

[29] A. Chattopadhyay and Z. Zilic, "GALDS: A Complete Framework for Designing Multiclock ASIC's and SoCs," IEEE Transactions on Very Large Scale Integration System, vol. 13, no. 6, 2005, pp. 641–654, Jun.

[30] E. Nilsson and J. Öberg, “Reducing Power And Latency In 2-D Mesh NoCs Using Globally Pseudo-asynchronous Locally Synchronous Clocking,” in Proc. of Int’l Conf. on Hardware Software Co-design, pp. 176-181, 2004.

[31] I. Soderquist, “Globally Updated Mesochronous Design Style,” IEEE Journal of Solid-State Circuits, vol. 38, no. 7, 2003, pp. 1242 - 1249, July.

[32] C. Chan, Y. Chang, H. Ho, and H. Chiueh, “A Thermal-Aware Power Management Soft-IP for Platform-Based SoC designs,” in Proc. of Int’l Symposium on System-on-Chip, pp. 181 - 184, Nov. 2004.

[33] J. Clabes, J. Friedrich, M. Sweet, J. Dilullo, et al., “Design and Implementation of the POWER5 Microprocessor,” in Proc. of Int’l Solid-State Circuits Conf., pp. 56-57, Jan. 2004.

[34] L. Shang, L.-S. Peh, A. Kumar, and N. K. Jha, “Thermal Modeling, Characterization and Management of On-Chip Networks,” in Proc. of Int’l Symp. On Microarchitecture, Dec. 2004.

[35] S. Borkar, “Performance, Power and the Platform,” Technology @ Intel Magazine, Nov. 2005.

[36] L. Kriaa, A. Bouchhima, M. Gligor, A.-M. Fouillart, F. Pétrot, and A.-A. Jerraya, “Parallel Programming of Multi-processor SoC: A HW–SW Interface Perspective,” International Journal of Parallel Programming, 2007.

[37] J.-M. Lu, H.-L. Wu, T.-M. Chiang, and W.-F. Chen, “High Performance and Low-Power Dual-Core SoC Platform for Portable Multimedia Applications,” SoC Technology Journal, no. 2, 2005, pp. 36-45, May.

[38] K. V. Nedovodeev, “Multimedia Data Processing on Dual-core SoC Multicore-24,” in Proc. of Int’l Symp. On Consumer Electronics, pp. 1-6, June 2006.

[39] K. Kim, D. Kim, and C. Park, “Real-Time Scheduling in Heterogeneous Dual-Core Architectures,” in Proc. Of Int’l Conf. on Parallel and Distributed Systems, vol. 2, July 2006.

[40] H.-J. Stolberg, M. Berekovic, S. Moch, L. Friebe, M. B. Kulaczewski, et al., “HiBRID-SoC: A Multi-Core SoC Architecture for Multimedia Signal Processing,” The Journal of VLSI Signal Processing, vol. 41, no. 1, 2005, pp. 9 – 20, August.

[41] H.-J. Stolberg, S. Moch, L. Friebe, A. Dehnhardt, M. B. Kulaczewski, M. Berekovic, P. Pirsch, “An SoC with Two Multimedia DSPs and a RISC Core for Video Compression Applications,” in Proc. of Int’l Solid-State Circuits Conf., pp. 330 - 531, Feb. 2004.

[42] H. Yue, Z. Wang, and K. Dai, “A Heterogeneous Embedded MPSoC for Multimedia Applications,” in Proc. of Int’l Conf. on High Performance Computing and Communications, pp. 591-600, Sep. 2006.

[43] P.-C. Tseng, C.-T. Huang, and L.-G. Chen. “Reconfigurable Discrete Wavelet Transform Processor for Heterogeneous Reconfigurable Multimedia Systems,” The Journal of VLSI Signal Processing, vol. 41, no. 1, 2005, pp. 35 - 47, August.

[44] F. J. Veredas, M. Scheppler, W. Moffat, and B. Mei, “Custom Implementation of the Coarse-Grained Reconfigurable ADRES Architecture for Multimedia Purposes,” in Proc. of Int’l Conf. on Field Programmable Logic and Applications, pp. 106-111, Aug. 2005.

[45] E. Nilsson and J. Öberg, “Reducing Power And Latency In 2-D Mesh NoCs Using Globally Pseudo-asynchronous Locally Synchronous Clocking,” in Proc. of Int’l Conf. on Hardware Software Codesign, pp. 176-181, 2004.

[46] I. Soderquist, “Globally Updated Mesochronous Design Style,” IEEE Journal of Solid-State Circuits, vol. 38, no. 7, 2003, pp. 1242 - 1249, July.

[47] C.-W. Wu, “SoC Testing Methodology and Practice,” in Proc. of Design, Automation and Test in Europe, pp. 1120-1121, 2005.

[48] Jason G. Tong, Mohammed A. S. Khalid, “Profiling tools for FPGA-based embedded systems: survey and quantitative comparison,” Journal of Computers, vol. 3, No. 6, 1-14, June 2008.

[49] G. Stitt, R. Lysecky, F. Vahid, “Dynamic hardware/software partitioning: a first approach,” Proceedings of the 40th conference on Design Automation. Pp250-255. June, 2003.

[50] L. Shannon, P. Chow, “Using Reconfigurability to Achieve Real-Time Profiling for Hardware/Software Co-design,” Proc. Of the 12th International Symposium on Field Programmable Gate Arrays. Pp 190-199. February 2004.

[51] R. Lysecky, F.Vahid, “A Study of the Speedups and Competitiveness of FPGA Soft Processor Cores using Dynamic Hardware/Software Partitioning,”

Proceedings of the Conference on Design, Automation and Test in Europe. Pp 170-173. March, 2005.

[52] J. G. Tong, M. A. S. Khalid, “Profiling CAD Tools: A Proposed Classification. Proceeding of the 19th International Conference on Microelectronics,” Pp.253-256. December, 2007.

[53] P. Pop, P. Elese, and Z. Peng, “Analysis and Synthesis of Distributed Real-Time Embedded Systems,” 1st ed. The Netherlands: Kluwer Academic Publishers, 2004.

[54] R. Lysecky, S. Cotterell, and F. Vahid, “A Fast On-Chip Profiler Memory,” in Proc. of the 39th Conference on Design Automation, pp. 28–33. June 2002.

[55] J. Fenlason and R. Stallman, GNU gprof, January 1997. [Online]. Available: http://www.gnu.org/software/binutils/manual/gprof-2.9.1/.

[56] The Linux Homepage. [Online]. Available: http://www.linux.org.

[57] The UNIX System. [Online]. Available: http://www.unix.org.

[58] D. A. Varley, “Practical Experience of the Limitations of gprof,” in Software Practice and Experience, 1993, pp. 461–463.

[59] Using UltraSPARC-IIICu Performance Counters to Improve Application Performance. Sun Microsystems. [Online]. Available: http://developers.sun.com/prodtech/cc/articles/pcounters.html.

[60] Intel Corporation, IA-32 Intel Architecture Software Developer’s Manual, Accessed February 2006. [Online]. Available: http://developers.sun.com/prodtech/cc/articles/pcounters.html.

[61] AMD Athlon Processor, “x86 Code Optimization Guide,” 2002.

[62] M. Itzkowitz, J. N. W. Brian, C. Aoki, and N. Kosche, “Memory profiling using hardware counters,” in Proc. Of the 2003 ACM/IEEE conference on Supercomputing, July 2003, pp. 17–30.

[63] N. Ohba and K. Takano, “An SoC Design Methodology using FPGAs and Embedded Processors,” in Proc. of the 41st Annual Conference on Design Automation, June 2004, pp. 747–752.

[64] Altera Corporation. [Online]. Available: http://www.altera.com.

[65] Xilinx Incorporated. [Online]. Available: http://www.xilinx.com.

[66] "Nios II Processor Handbook," Altera Corporation, 2011.

[67] "MicroBlaze Processor Reference Guide," Xilinx Incorporated, 2011.

[68] Blank Tom, "A survey of Hardware Accelerators used in Computer-Aided Design," Design & Test of Computers IEEE. vol. 1, Issue 3. Pp 21-39. Aug, 1984.

[69] Newby, G. B., "Hardware acceleration prospects and challenges for high performance computing," Computer Systems and Applications. IEEE/ACS International Conference. Pp 841-844. May, 2009.

[70] K. Gulati, S. P Khatri, "Hardware acceleration of EDA algorithms: custom ICs, FPGAs and GPUs," New York; London; Springer, 2010.

[71] N. Nedjah, L. de Macedo Mourelle, "Co-design for system acceleration: a quantitative approach," Dordrecht; London: Springer, 2007.

[72] M. Lin, "Digital system designs and practices: using Verilog HDL and FPGAs," Singapore; Hoboken, NJ. J. Wiley & Sons, 2008.

[73] P. J Ashenden, "The designer's guide to VHDL," San Francisco, CA. 2002.

[74] Iain E. G. Richardson, "H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia," John Wiley & Sons, Ltd. 2003.

[75] Anderson, M. "VCR quality video at 1.5 Mbits/s," National Communication Forum, Chicago, Oct. 1990.

[76] ITU-T and ISO/IEC JTC 1, "Generic coding of moving pictures and associated audio information – Part 2: Video," ITU-T Recommendation H.262 – ISO/IEC 13818-2 (MPEG-2), Nov. 1994.

[77] "Coding of moving pictures and associated audio," Committee Draft of Standard ISO11172: ISO/MPEG 90/176, Dec. 1990.

[78] Benson, K. Blair, and Donald G. Fink, "HDTV--Advanced Television for the 1990s," New York: Intertext Publications: McGraw-Hill Pub. Co., 1991.

[79] Joint Video Team of ITU-T and ISO/IEC JTC 1, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 ISO/IEC 14496-10 AVC)," Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVT-G050, March 2003.

[80] ITU-T, "Video Codec for Audiovisual Services at px64 kbit/s," ITU-T Recommendation H.261, Version 1: Nov. 1990; Version 2: Mar. 1993.

[81] ITU-T, “Video coding for low bit rate communication,” ITUT Recommendation H.263; version 1, Nov. 1995; version 2, Jan. 1998; version 3, Nov. 2000.

[82] Wiegand, T., Sullivan, G.J., Bjontegaard, G., Luthra, A, “Overview of the H.264/AVC video coding standard,” IEEE Transactions on Circuits and Systems for Video Technology, vol.13, pp. 560-576, July, 2003.

[83] Ostermann, J., Bormans, J., List, P., Marp, D., Narroschke, M., Pereira, F., Stockhammer, T., Wedi, T., “Video coding with H.264/AVC: tools, performance, and complexity,” Circuits and Systems Magazine, vol.4, pp.7-28, 2004.

[84] T. Wedi, “Motion Compensation in H.264/AVC,” in IEEE Transactions on Circuits and Systems for Video Technology, this issue.

[85] T. Wiegand, X. Zhang, and B. Girod, “Long-Term Memory Motion-Compensated Prediction,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 9, no. 1, pp. 70-84, Feb. 1999.

[86] T. Wiegand and B. Girod, “Multi-frame Motion- Compensated Prediction for Video Transmission,” Kluwer Academic Publishers, Sept. 2001.

[87] M. Flierl, T. Wiegand, and B. Girod, “A Locally Optimal Design Algorithm for Block-Based Multi-Hypothesis Motion-Compensated Prediction,” in Data Compression Conference, Snowbird, USA, Mar. 1998, pp. 239-248.

[88] D. Marpe, H. Schwarz, and T. Wiegand: “Context-Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard,” in IEEE Transactions on Circuits and Systems for Video Technology, this issue.

[89] Steve Kilts. “Advanced FPGA Design: Architecture, Implementation, and Optimization,” John Wiley & Sons, 2007.

[90] ML505/ML506/ML507 Evaluation Platform User Guide. [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf.

[91] Intel Corporation, “Using Intel VTune’s Counter Monitor,” January 2005.

[92] P. Giusto, G. Martin, and E. Harcourt, “Reliable estimation of execution time of embedded software,” in Proc. of Design Automation and Test in Europe, pp. 580 – 588, 2001.

[93] L. Shannon, Paul Chow, “Using Reconfigurability to Achieve Real-time Profiling for Hardware/Software Codesign,” in Proc. of 12th International Symposium of Field programmable gate arrays, pp. 190 – 199, 2004.

[94] R. Duarte, C. Liu, and X. Niu, “RSA Cryptography Acceleration for Embedded System,” The 6th International Workshop on Unique Chips and Systems (UCAS-6), in conjunction with MICRO-43, Atlanta, GA, December 4, 2010.

[95] J. Villarreal, D. Suresh, G. Stitt, F. Vahid et al., “Improving Software Performance with Configurable Logic,” Kluwer Journal on Design Automation of Embedded Systems, Vol. 7, No. 4, pp.325 -339, November 2002.

[96] D. C. Suresh, W. A. Najjar, F. Vahid, et al., “Profiling Tools for Hardware/Software Partitioning of Embedded Applications,” in Proc. of Language, Compiler, and Tool for Embedded Systems, Vol. 38, Issue. 7, pp. 189-198, 2003.

[97] Tung-Chien Chen, Yu-Wen Huang, Liang-Gee Chen; “Analysis and design of macroblock pipelining for H.264/AVC VLSI architecture,” Proceedings of International Symposium on Circuits and Systems, vol. 2, pp.273-276, May, 2004.

[98] Roman C. Kordasiewicz, Shahram Shirani; “ASIC and FPGA implementations of H.264 DCT and quantization blocks,” ICIP, vol.3, pp.1020-1023, Sep, 2005.

[99] Elgato website: [Online]. Available: http://www.elgato.com/elgato/na/mainmenu/products/Turbo264HD/product1.en.html.

[100] H. C. Lin, Y. J. Wang, K. T. Cheng, et al., “Algorithms and DSP Implementation of H.264/AVC,” Design Automation, 24 – 27, Jan. 2006.

[101] H.264/MPEG-4 AVC [Online]. Available: http://en.wikipedia.org/wiki/H264.

[102] Introduction of YUV [Online]. Available: http://en.wikipedia.org/wiki/Yuv.

[103] K.J. Horadam, “Hadamard Matrices and Their Applications,” Princeton university press, ISBN: 9780691119212, 2006.

[104] A.N. Akansu and R. Poluri, “Walsh-Like Nonlinear Phase Orthogonal Codes for Direct Sequence CDMA Communications,” IEEE Trans. on Signal Processing, vol. 55, pp. 3800–3806, July 2007.

[105] Li-Jun Yan, Jeng-Shyang Pan. “Generalized Discrete Fractional Hadamard Transformation and its Application on the Image Encryption,” Intelligent Information Hiding and Multimedia Signal Processing, IIHMSP, Vol. 1, pp 457-460, Nov.2007.

[106] Virtex-5 FPGA Family. [Online]. Available: http://www.xilinx.com/products/virtex5/.

[107] Introduction of Synplify pro synthesis tool. [Online]. Available: http://www.synopsys.com/Tools/Implementation/FPGAImplementation/CapsuleModule/syn_pro_ds.pdf.

[108] Introduction of Xilinx Local Memory Bus. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/lmb_v10.pdf.

[109] Introduction of Xilinx PLB bus. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf.

[110] LogiCORE IP Fast Simplex Link (FSL) V20 Bus. [Online.] Available: http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf.

[111] Xilinx Xpower Analyzer. [Online]. Available: http://www.xilinx.com/products/design_tools/logic_design/verification/xpower_an.htm.

[112] Modelsim software. [Online]. Available: http://www.mentor.com/products/fv/modelsim/.

[113] R.C. Gonzalez, R.E. Woods, "Digital Image Processing (2nd)" November 9, 2001, Prentice Hall, ISBN: 9781846283796/1846283795.

[114] S. Halder, D. Bhattacharjee, et al., "A Fast FPGA Based Architecture for Sobel Edge Detection," in Proc. of Progress in 16th VLSI Design and Test, pp. 300-306, July, 2012.

[115] T. A. Abbasi, M. U. Abbasi, "A novel FPGA-based architecture for Sobel edge detection operator," International Journal of Electronics, vol. 94, Issue 9, pp. 889 – 896, 2007.

[116] C. Pradabpet, N. Ravinu, et al., "An Efficient Filter Structure for Multiplierless Sobel Edge Detection," in Proc. of Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA 2009), pp. 40 – 44, July, 2009.

[117] Z. E. M. Osman, F. A. Hussin et al., "Hardware Implementation of an Optimized Processor Architecture for Sobel Image Edge Detection Operator," in Proc. of Intelligent and Advanced Systems (ICIAS), pp. 1 – 4, June, 2010.

[118] V. Sanduja, R. Patial, “Sobel Edge Detection using Parallel Architecture based on FPGA,” International Journal of Applied Information Systems, vol. 4, No. 4, pp. 20 – 24, July, 2012.

[119] AMBA specifications. [Online]. Available: http://www.arm.com/products/system-ip/amba/amba-open-specifications.php.

[120] X. Niu, Jeffrey Fan, “System Verification of Hardware Optimization based on Edge Detection,” Journal of Circuits and Systems (CS), Vol.4, No.3, July issue, 2013.

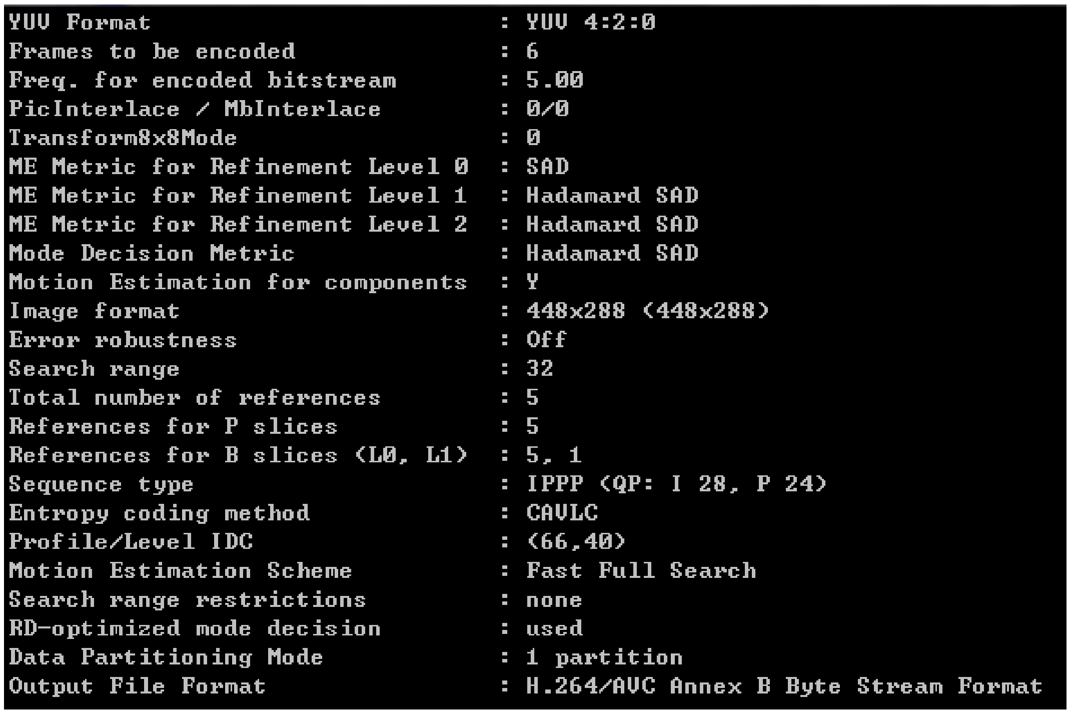
[121] X. Niu, Jeffrey Fan, “System-on-a-Chip (SoC) based Hardware Acceleration for Video Codec,” International Symposium on Photonics and Optoelectronics 2013, (to appear on Optics and Photonics Journal (OPJ) May 2013 issue).

[122] X. Niu, L. Galarza, Y. Gao, Jeffrey Fan, “Software-hardware co-design for video coding acceleration,” 44th IEEE Southeastern Symposium on System Theory (SSST'12), Jacksonville, FL, pp. 57-60, March 11-13, 2012.

[123] X. Niu, N. Nartasilpa, C. Chaiyanan, Jeffrey Fan, “Low power mobile operating systems with profiling hardware acceleration,” i-manager's Journal on Software Engineering (IJSE), Vol. 5, No. 2, pp. 16-25, Jan-Mar issue, 2011.

APPENDICES

APPENDIX A – H.264 ENCODING PROCESS



```
YUV Format                          : YUV 4:2:0
Frames to be encoded                : 6
Freq. for encoded bitstream         : 5.00
PicInterlace / MbInterlace          : 0/0
Transform8x8Mode                    : 0
ME Metric for Refinement Level 0    : SAD
ME Metric for Refinement Level 1    : Hadamard SAD
ME Metric for Refinement Level 2    : Hadamard SAD
Mode Decision Metric                : Hadamard SAD
Motion Estimation for components    : Y
Image format                        : 448x288 (448x288)
Error robustness                    : Off
Search range                        : 32
Total number of references          : 5
References for P slices             : 5
References for B slices (L0, L1)    : 5, 1
Sequence type                       : IPPP (QP: I 28, P 24)
Entropy coding method               : CAVLC
Profile/Level IDC                   : (66,40)
Motion Estimation Scheme            : Fast Full Search
Search range restrictions           : none
RD-optimized mode decision          : used
Data Partitioning Mode              : 1 partition
Output File Format                  : H.264/AVC Annex B Byte Stream Format
```

APPENDIX B – HARDWARE GATE LEVEL MAP

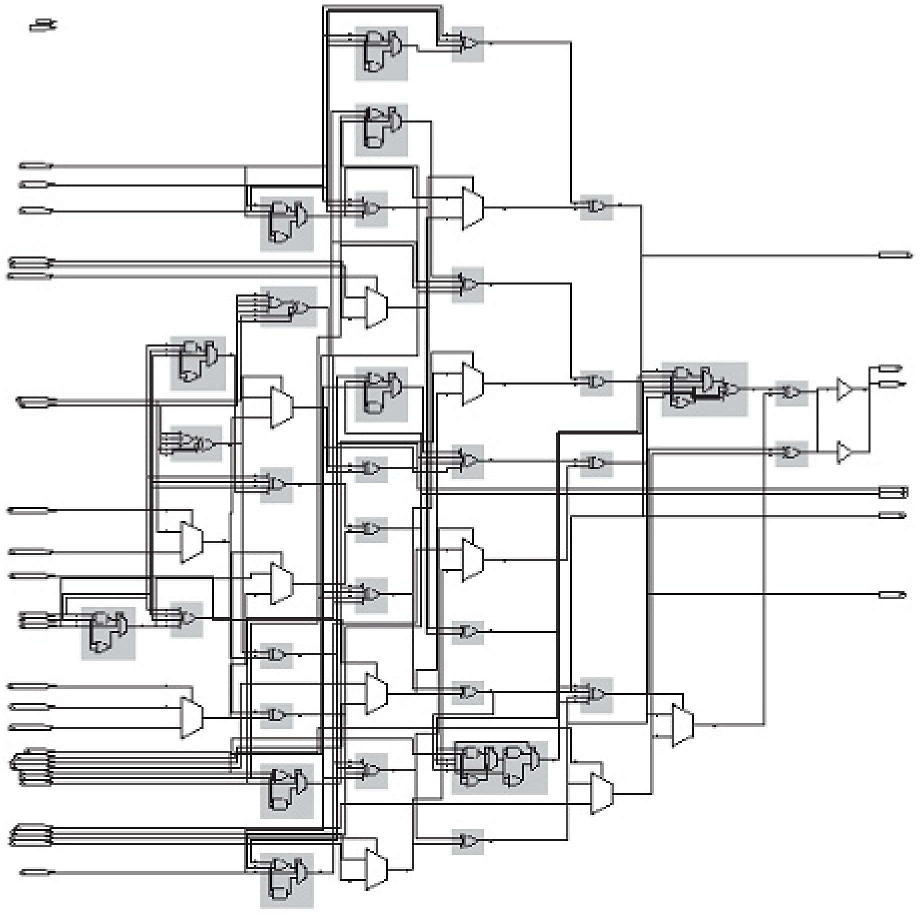


Figure B-1 Hadamard_1 Gate Level Map

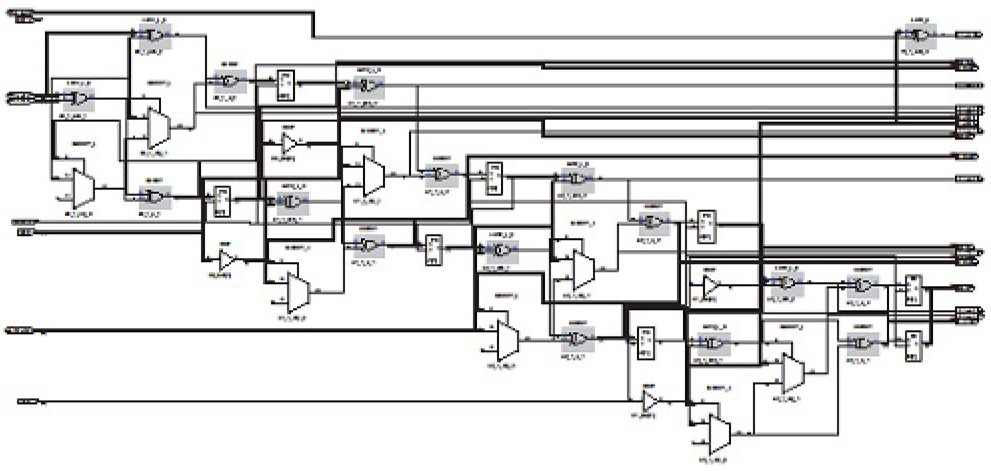


Figure B-2 Hadamard_2 Gate Level Map

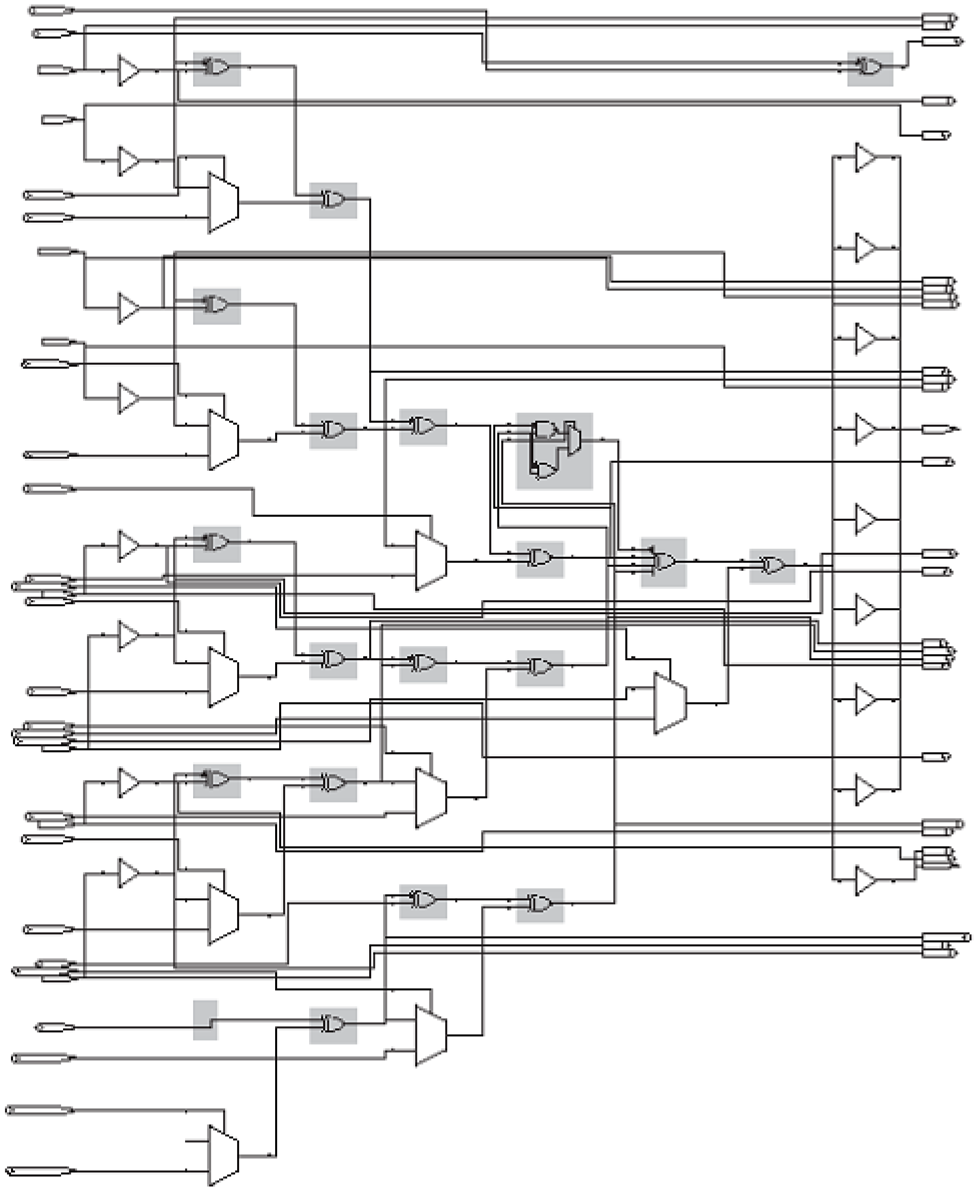


Figure B-3 Hadamard_3 Gate Level Map

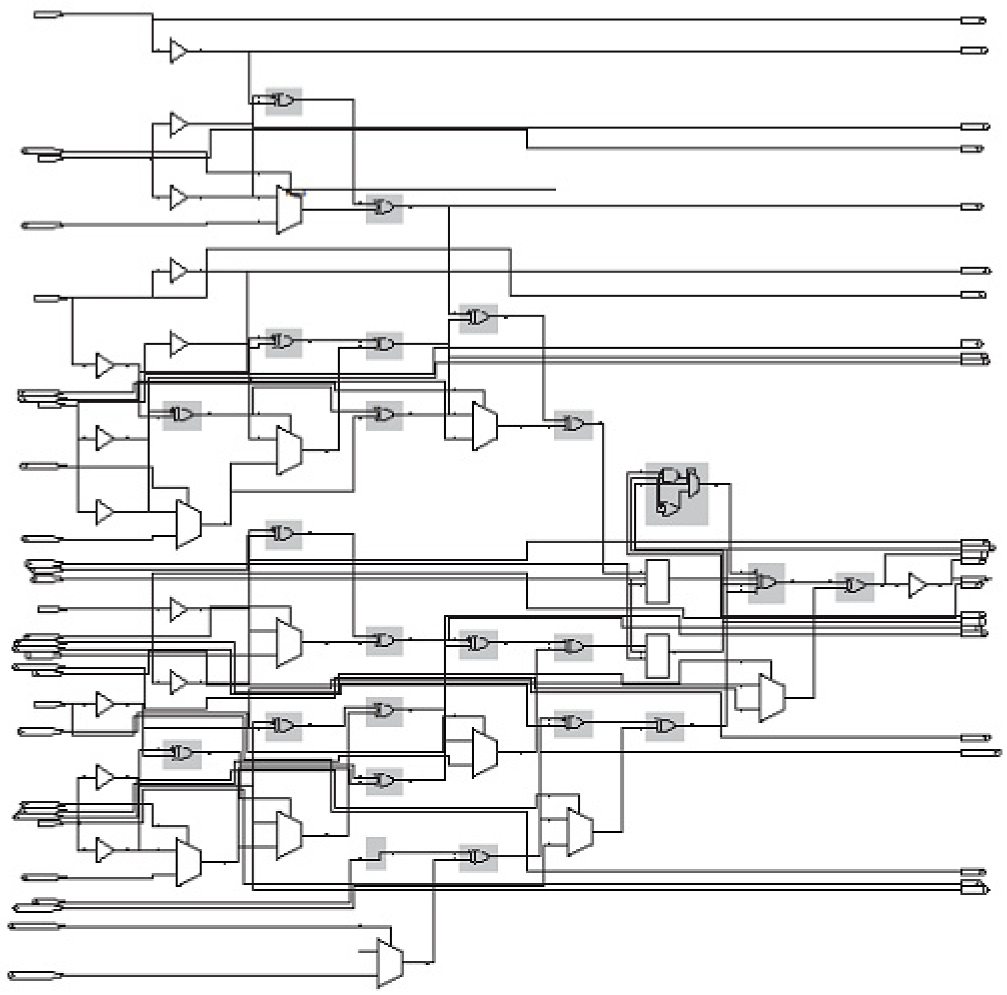


Figure B-4 Hadamard_4 Gate Level Map

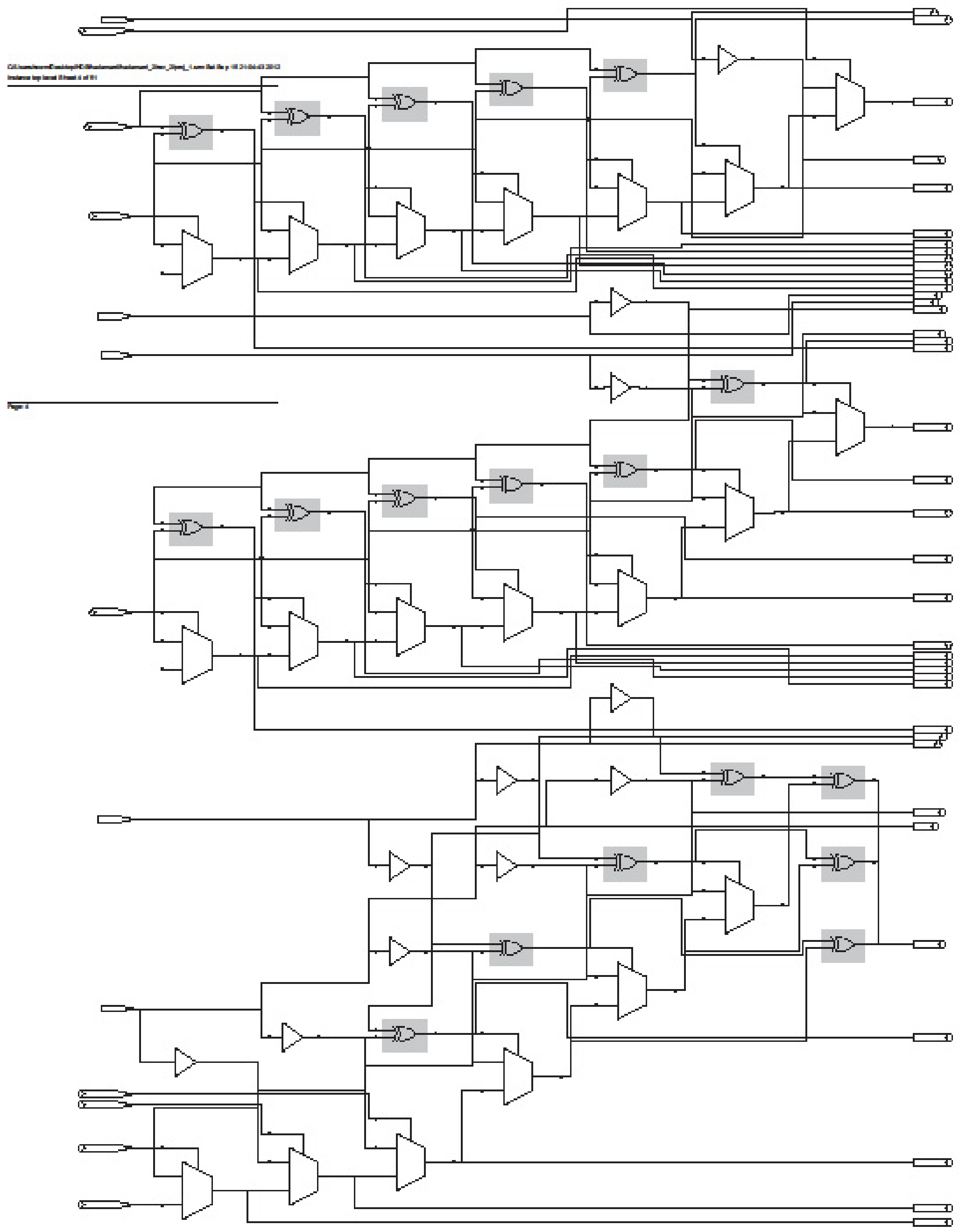


Figure B-5 Hadamard_5 Gate Level Map

VITA

Xinwei Niu

2001-2005	B.S., Electrical Engineering Xi'an University of Architecture and Technology Xi’an, Shaanxi, China P. R.
2005-2008	M.S., Signal and Information Processing Northwestern Polytechnical University Xi’an, Shaanxi, China P.R.
2008-2012	Doctorate Candidate in Electrical Engineering Florida International University Miami, Florida

PUBLICATIONS

Xinwei Niu, Jeffrey Fan, “System Verification of Hardware Optimization based on Edge Detection,” Journal of Circuits and Systems (CS), Vol.4, No.3, July issue, 2013.

Xinwei Niu, Jeffrey Fan, “System-on-a-Chip (SoC) based Hardware Acceleration for Video Codec,” International Symposium on Photonics and Optoelectronics 2013, (to appear on Optics and Photonics Journal (OPJ) May issue).

X. Yang, **Xinwei Niu**, Jeffrey Fan, “Mixed-Signal System-on-a-Chip (SoC) Verification Based on System Verilog Real Data Model,” 45th IEEE Southeastern Symposium on System Theory (SSST’13), Waco, TX, March 11, 2013 (Accepted).

Xinwei Niu, L. Galarza, Y. Gao, Jeffrey Fan, “Software-hardware co-design for video coding acceleration,” 44th IEEE Southeastern Symposium on System Theory (SSST'12), Jacksonville, FL, pp. 57-60, March 11-13, 2012.

Xinwei Niu, N. Nartasilpa, C. Chaiyanan, Jeffrey Fan, “Low power mobile operating systems with profiling hardware acceleration,” i-manager's Journal on Software Engineering (IJSE), Vol. 5, No. 2, pp. 16-25, Jan-Mar issue, 2011.

R. Duarte, C. Liu, **Xinwei Niu**, “RSA Cryptography Acceleration for Embedded System,” The 6th International Workshop on Unique Chips and Systems (UCAS-6), in conjunction with MICRO-43, Atlanta, GA, December 4, 2010.

L. Weng, **Xinwei Niu**, C. Liu, N. Pissinou, “Intelligent Controller with Cache for Critical Infrastructures,” The IET International Conference on Frontier Computing – Theory, Technologies and Applications (IET FC 2010), Taichung, Taiwan, pp. 60-65, August, 2010.

Xinwei Niu, H. Li, C. Zhang, J. Ma, “Research on Reconfigurable FPGA based on Image Fusion,” Information Security and Communications Privacy, No.5, pp.67-70, 2008.

H. Li, R. Xu, **Xinwei Niu**, S. Xie, “Application of NiosⅡ System in the Color Ultrasonic Diagnostic System,” Microprocessors, No. 3, pp. 89-92, 2007.

H. Li, C. Qu, B. Zhu, **Xinwei Niu**, “A Novel Semi Blind Color Image Watering Algorithm Based on DWT,” China Information Security, No. 3, pp. 92-93, 2007.

J. Liu, H. Li, **Xinwei Niu**, “The Algorithm of Handwritten Chinese Character Recognition Based on Multiple MHMM,” China Information Security, No. 2, pp. 75-77, 2007.