

國立交通大學

電信工程學系

碩士論文

具多齊質性處理器核心之多媒體串流處理架構



Media Streaming Architecture with  
Homogeneous Processor Cores

研究生：劉嘉儀

指導教授：闕河鳴博士

中華民國九十六年七月

具多齊質性處理器核心之  
多媒體串流處理架構

Media Streaming Architecture with  
Homogeneous Processor Cores

研究生：劉嘉儀

Student : Chia-Yi Liou

指導教授：闕河鳴 博士

Advisor : Dr. Herming Chiueh



A Thesis

Submitted to Department of Communication Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Communication Engineering

July 2007

Hsinchu, Taiwan.

中華民國九十六年七月

# 具多齊質性處理器核心之 多媒體串流處理架構

研究生：劉嘉儀

指導教授：闕河鳴 博士

國立交通大學

電信工程學系碩士班

## 摘要



隨著科技的發展與進步，在現今的生活中應用於多媒體運算之可攜式嵌入式系統的重要性與日俱增。然而由於傳統運算模型中之記憶體存取模型與處理核心和記憶體間的效能間隙落差，導致多媒體運算無法有效率的對應並且實現在傳統的處理器架構模型上。另外在硬體實現的系統架構上亦是產生效能無法提升的重要因素之一。因此所提出的多媒體處理架構採用史丹佛大學提出之串流處理模型配合上多種硬體實現的系統架構來克服傳統處理器架構所造成效率低落。並且提供一具高平行度和有效率運算速率的多媒體運算平台。

在本論文中，設計並下線製作一個與 AMBA 介面相容之多媒體處理單元作為構成具多齊質性處理核心之多媒體串流處理架構的核心運算單元。除此之外亦設計實做浮點運算處理器，利用此浮點運算處理器提供此一與 AMBA 介面相容之多媒體處理單元有效率的浮點運算處理能力，使其可以更廣泛的應用於各種多媒體處理運算中。透過不同運算單元與架構間的效能評估與比較，證實了僅需要

些許的硬體成本即可提供更有效率且更廣泛的多媒體運算處理能力。另外此效能評估與比較亦證實了具多齊質性處理核心之多媒體串流處理架構在擁有不同數目之處理核心時，在合理的硬體成本之下其效能可以有效的提升。



# Media Streaming Architecture with Homogeneous Processor Cores

Student: Chia-Yi Liou

Advisor: Dr. Herming Chiueh

Department of Communication Engineering

National Chiao Tung University

Hsinchu, Taiwan

## Abstract

As the evolution of information technology, embedded systems with media applications for portable devices are more and more important in modern life. However, the conventional processor architecture does not handle the processing requirement of media applications very well since the characteristics of media applications and other inheritance disability from conventional microprocessor architecture's memory accessing model and processor-memory performance gap.

Recent research shows that the stream processing model and stream processor architecture are suitable for media applications. However, software implementations for a streaming processor are not a trivial job since it evolves a lot of hand and manual optimization in memory exchange and tread deployment to different processor element or functional unit.

In this thesis, a processing element for reconfigurable homogenous ALU cluster and its Advanced Microcontroller Bus Architecture (AMBA) platform interface has been designed and implemented. The proposed design integrated platform based design methodology and stream processing model to overcome the challenge of media applications. The proposed homogenous ALU cluster is utilized as a reconfigurable hardware accelerator for specific and different functions in media applications. The chosen AMBA interface provides an integration platform for embedded operating system and programming development environment. The combination of these methodologies provides a turnkey solution for media applications development in modern portable devices.

The ALU cluster IP with AMBA interface is taped out using TSMC 0.15um technology and operates at 100MHz. The chip area is  $3.9*3.9 \text{ mm}^2$  and gate count is 0.2 million. A 4-layer FRP printed circuit board is designed and fabricated as the daughter card for system integration. The daughter card carries the designed chip is integrated to ARM versatile platform board as the system integration and application development environment. In addition, a floating point operation unit for ALU cluster IP is proposed and implemented and it will be integrated with ALU cluster IP as the future revision of the hardware accelerator. The hard macro of the floating point unit operates at 75MHz, its area and gate count is  $0.415\text{mm}^2$  and 0.02 million respectively. The performance evaluation and comparison in floating point operation benchmark between different proposed architectures are presented. Media applications can be developed for proposed reconfigurable homogenous processing elements in the future using the chips and systems build in this thesis.



# *Acknowledgement*

本篇碩士論文得以順利完成，首先要感謝我的指導教授 闕河鳴博士。老師擁有淵博的學識，總能讓學生在研究遇到瓶頸的時候給予寶貴的指導及建議，使我能夠突破與進步。更由於老師在平日培養學生獨立思考與分析、解決問題的能力，使我對於研究的領域建立正確的態度及觀念。

再者要感謝晶片系統設計實驗室的學長姐、同學以及學弟妹們，在我的研究上及生活上給與諸多的指教及幫助。因為你們的支持，才讓我得以擁有一段快樂充實的碩士班研究生活，謝謝大家。

最後，我要感謝父母的栽培與養育之恩、所有關心我的家人與朋友、以及我生長的這塊土地、社會與國家。

誠心感謝並祝福所有提攜幫助我和支持鼓勵我的大家，謝謝各位並祝福各位。

劉嘉儀

# CONTENTS

中文摘要

English Abstract.....

Acknowledgment.....

Content.....

List of Tables.....

List of Figures.....

<b>Chapter 1 Introduction.....</b>	<b>1</b>
1.1 Motivation.....	1
1.2 Organization.....	3
<b>Chapter 2 Background and Challenges.....</b>	<b>4</b>
2.1 Issues of Programming Model.....	4
2.2 Issues of System Architecture.....	6
<b>Chapter 3 Development Roadmap and Proposed Design.....</b>	<b>10</b>
3.1 Developmental Roadmap.....	11
3.1.1 Motivation.....	11
3.1.2 Roadmap.....	11
3.1.2.1 Stream Programming Model.....	11
3.1.2.2 Developmental Roadmap.....	14
3.2 An ALU cluster.....	17
3.2.1 Micro-Architecture of an ALU cluster.....	17
3.3 An ALU cluster Intellectual Property.....	18
3.3.1 Overview of AMBA.....	18



3.3.1.1	Introduction of AMBA AHB.....	20
3.3.1.2	Bus Interconnection.....	21
3.3.1.3	Signals for the protocol of AMBA AHB slave.....	22
3.3.1.4	Basic Transfer.....	24
3.3.1.5	Transfer Type.....	25
3.3.1.6	Address Decoding.....	26
3.3.1.7	Burst Operation.....	27
3.3.2	Micro-Architecture of an ALU cluster Intellectual Property.....	28
3.4	Floating Point Units for the ALU cluster IP.....	32
3.4.1	Design Consideration.....	33
 <b>Chapter 4 Implementation Results and Performance Evaluation.....</b>		<b>37</b>
4.1	Implementation and Testing Results of An ALU cluster.....	38
4.2	Verification and Implementation Results of An ALU cluster Intellectual Property.....	43
4.2.1	An ALU cluster IP with Magnetic RAM.....	43
4.2.1.1	Introduction of Magnetic RAM.....	44
4.2.1.2	Modified ALU cluster IP for Magnetic RAM.....	44
4.2.2	Implementation Results.....	45
4.2.3	Circuit Verification.....	50
4.2.4	Chip Testing.....	55
4.3	Circuit Implementation and Results of Floating Point Units for the ALU cluster IP.....	57
4.4	Performance Evaluation and Comparison.....	62
4.4.1	Selected Benchmark.....	63
4.4.2	Evaluation and Comparison Results.....	64
 <b>Chapter 5 Conclusion and Future Work.....</b>		<b>80</b>
5.1	Conclusion.....	80
5.2	Future Work.....	81
 <b>Bibliography.....</b>		<b>82</b>

# LIST of TABLES

<b>Table 2.1</b>	System Architecture vs. media application.....	<b>9</b>
<b>Table 3.1</b>	Comparison between programming models.....	<b>14</b>
<b>Table 3.2</b>	Burst Signal Encoding.....	<b>27</b>
<b>Table 3.3</b>	Active Byte Lanes for a 32 bits big endian data bus.....	<b>28</b>
<b>Table 3.4</b>	Active Byte Lanes for a 32 bits little endian data bus.....	<b>28</b>
<b>Table 3.5</b>	Format of single and double precision IEEE 754 floating point number.....	<b>34</b>
<b>Table 3.6</b>	Effective Range of the IEEE 754 floating point number.....	<b>34</b>
<b>Table 4.1</b>	Implementation Results Summary of ALU cluster.....	<b>38</b>
<b>Table 4.2</b>	Testing results summaries of ALU cluster.....	<b>43</b>
<b>Table 4.3</b>	Summary of Implementation Characteristics.....	<b>46</b>
<b>Table 4.4</b>	The Definitions of I/O the ports.....	<b>49</b>
<b>Table 4.5</b>	Summary of the Implementation Results.....	<b>59</b>
<b>Table 4.6</b>	Performance Evaluation Results for Original Integer Architecture.....	<b>66</b>
<b>Table 4.7</b>	Performance Evaluation Results for Floating Point Unit and Original Integer Architecture Mixed.....	<b>67</b>
<b>Table 4.8</b>	Performance Evaluation Results in Execution Time.....	<b>71</b>

# LIST OF FIGURES

<b>Fig 2.1</b>	Conventional Programming Model.....	5
<b>Fig 2.2</b>	Processor-Memory Performance Gap.....	5
<b>Fig 2.3</b>	Application Specified Integrated Circuit Design.....	6
<b>Fig 2.4</b>	An Example of Platform-Based Architecture.....	7
<b>Fig 2.5</b>	A Diagram of Reconfigurable Architecture.....	8
<b>Fig 3.1</b>	Stream Programming model.....	12
<b>Fig 3.2</b>	Example of 1024-points radix-2 Fast Fourier Transform.....	13
<b>Fig 3.3</b>	Example of Stereo Depth Extraction.....	13
<b>Fig 3.4</b>	Development roadmap.....	15
<b>Fig 3.5</b>	Micro-Architecture of ALU cluster.....	17
<b>Fig 3.6</b>	Diagram of AMBA-based system.....	20
<b>Fig 3.7</b>	Diagram of AMBA AHB interconnection.....	22
<b>Fig 3.8</b>	Diagram of AHB slave interface.....	23
<b>Fig 3.9</b>	An example of simple transfer.....	24
<b>Fig 3.10</b>	The example of the transfer extended.....	25
<b>Fig 3.11</b>	Slave Selected Signal.....	26
<b>Fig 3.12</b>	The Proposed ALU Cluster IP Architecture.....	29
<b>Fig 3.13</b>	The state diagram of the finite state machine.....	31
<b>Fig 3.14</b>	An ALU cluster IP with Floating Point Unit Supported Architecture.....	36
<b>Fig 4.1</b>	Physical Layout of an ALU cluster.....	39
<b>Fig 4.2</b>	Floorplan and Pad Assignment of an ALU cluster.....	40
<b>Fig 4.3</b>	Microphotograph of taped out ALU cluster.....	41
<b>Fig 4.4</b>	An ALU cluster with CQFP128 package.....	41
<b>Fig 4.5</b>	An ALU cluster with PCB board.....	42
<b>Fig 4.6</b>	Modified ALU cluster IP architecture for MRAM.....	45
<b>Fig 4.7</b>	Physical Layout of an ALU Cluster IP.....	46
<b>Fig 4.8</b>	Pads Assignment of an ALU Cluster IP.....	47
<b>Fig 4.9</b>	Die Microphotograph of Taped Out Chip.....	48
<b>Fig 4.10</b>	Photograph of Prototype with Package.....	48
<b>Fig 4.11</b>	The input function and coefficients of the FIR filter system...	51
<b>Fig 4.12</b>	Output results of the FIR filter system.....	52
<b>Fig 4.13(a)</b>	Post-Layout Simulation Results of an ALU cluster IP ( )...	53

<b>Fig 4.13(b)</b>	Post-Layout Simulation Results of an ALU cluster IP ( )...	<b>53</b>
<b>Fig 4.13(c)</b>	Post-Layout Simulation Results of an ALU cluster IP ( )...	<b>54</b>
<b>Fig 4.13(d)</b>	Post-Layout Simulation Results of an ALU cluster IP ( )...	<b>54</b>
<b>Fig 4.13(e)</b>	Post-Layout Simulation Results of an ALU cluster IP ( )...	<b>55</b>
<b>Fig 4.14</b>	The Printed Circuit Board (PCB) for the manufactured chip..	<b>55</b>
<b>Fig 4.15</b>	Testing Equipments – Logic Analyzer System.....	<b>56</b>
<b>Fig 4.16</b>	Connection between the Chip and Testing Equipments.....	<b>56</b>
<b>Fig 4.17</b>	Physical Layout of the Type 1 FPU macro.....	<b>58</b>
<b>Fig 4.18</b>	Physical Layout of the Type 2 FPU macro.....	<b>58</b>
<b>Fig 4.19</b>	Physical Layout of the Type 3 FPU macro.....	<b>59</b>
<b>Fig 4.20</b>	Full View of Post-Layout Simulation Results for Type 1 FPU.....	<b>60</b>
<b>Fig 4.21</b>	Interception of Post-Layout Simulation Results for Type 1 FPU.....	<b>60</b>
<b>Fig 4.22</b>	Full View of Post-Layout Simulation Results for Type 2 FPU.....	<b>61</b>
<b>Fig 4.23</b>	Interception of Post-Layout Simulation Results for Type 2 FPU.....	<b>61</b>
<b>Fig 4.24</b>	Full View of Post-Layout Simulation Results for Type 3 FPU.....	<b>62</b>
<b>Fig 4.25</b>	Interception of Post-Layout Simulation Results for Type 3 FPU.....	<b>62</b>
<b>Fig 4.26</b>	Flowchart of the length 32 Split-Radix FFT algorithm.....	<b>64</b>
<b>Fig 4.27</b>	Performance Evaluation of one cluster included in these architectures.....	<b>68</b>
<b>Fig 4.28</b>	Performance Evaluation of two clusters included in these architectures.....	<b>69</b>
<b>Fig 4.29</b>	Performance Evaluation of four clusters included in these architectures.....	<b>69</b>
<b>Fig 4.30</b>	Performance Evaluation of eight clusters included in these architectures.....	<b>70</b>
<b>Fig 4.31</b>	Performance Evaluation of one cluster included in execution time.....	<b>72</b>
<b>Fig 4.32</b>	Performance Evaluation of two clusters included in execution time.....	<b>73</b>
<b>Fig 4.33</b>	Performance Evaluation of four clusters included in execution time.....	<b>74</b>

<b>Fig 4.34</b>	Performance Evaluation of eight clusters included in execution time.....	<b>75</b>
<b>Fig 4.35</b>	Comparison of Performance Normalized in execution cycles	<b>76</b>
<b>Fig 4.36</b>	Comparison of Performance Normalized in execution time...	<b>76</b>
<b>Fig 4.37</b>	Performance Comparison for Different Number of Clusters used in cycles.....	<b>78</b>
<b>Fig 4.38</b>	Performance Comparison for Different Number of Clusters used in execution time.....	<b>78</b>
<b>Fig 5.1</b>	RealView Versatile Platform Baseboard for ARM926EJ-S...	<b>81</b>



# CHAPTER 1

## Introduction

### 1.1 Motivation

Portable systems are more and more important. They become essentials of our life. Furthermore, media applications are becoming a dominant portion of processing for portable entertainment system in modern life.

Multimedia processing applications are characterized by large available parallelism, little data reuse and high computation to memory access ratio. Large available parallelism due to each data stream is independent to others, so each stream is possible to be operated concurrently. The reason for the characteristic of little data reuse is that typical data reference in media applications require a single read and write per global data element. High computation to memory access ratio is needed because of large amount of data operations [1 - 4]. Thus these characteristics poorly match conventional general purpose processor architecture. The conventional programming model and processor architecture dealt with media applications traditionally are not efficient because of the characteristics of media applications, its memory accessing model and processor-memory performance gap [5].

In addition to programming models, system architectures used to implement the whole media processing system are also key factors to affect the efficiency of processing. System architectures such as application specific integrated circuits (ASIC), platform-based architecture and reconfigurable architecture are used to implement the hardware for media applications. However, these architectures for media processing have their own drawbacks separately. They suffer from lacking of flexibility, programmability, and inefficient communication bandwidth. The issues of programming model and system architecture mentioned above limit the processing requirement needed for modern media application in mobile system.

However, streaming programming model has been suggested as an efficient programming model for both media applications and base-band architecture for software defined radios [6] [7]. In order to build next generation media processing system, advantages from different system architectures are integrated. The pros of reconfigurable architecture and platform-based architecture will overcome the drawbacks of using above-mentioned architecture separately.

This thesis presented a processing element for reconfigurable homogenous ALU cluster and its Advanced Microcontroller Bus Architecture (AMBA) platform interface has been designed and implemented. The proposed design provides enough processing requirement for media applications and utilized as a reconfigurable hardware accelerator for specific and different functions in media applications. AMBA AHB interface in this design provides an integration platform for embedded operating system and programming development environment. The combination of these design methodologies will be a suitable solution in development applications for portable devices.

All design and verification of the proposed architecture are finished with cell-based design flow. The chip is taped out using TSMC 0.15um CMOS technology and operates at 100MHz. The die size and gate count are 15.2 mm<sup>2</sup> and 0.2 million respectively. Utilize COB (PGA256) as package material. The pad number of proposed chip is 130. The designed chip with daughter card will be integrated with ARM926EJ-S versatile baseboard to form a media streaming system and as the development environment for applications.

Furthermore, a floating point operation unit is critical in the majority of media application and makes the applications efficient. A floating point operation unit for ALU cluster IP is proposed and implemented in this thesis as a modified version of the hardware accelerator. The hard macro of three different type floating point unit operates at 75MHz, 25MHz and 25MHz, its area and gate count is 0.415 mm<sup>2</sup>, 0.529mm<sup>2</sup>, 0.396mm<sup>2</sup> and 0.02million, 0.03 million, 0.02 million respectively. Then the performance evaluation and comparison between two different target architectures are presented and shows the results that the floating point unit is efficient and critical for the proposed architecture.

## 1.2 Organization

In the beginning of Chapter 2, the background and challenges about this thesis are introduced. Issues of programming models and system architectures for media applications are discussed.

Next come the details of the development roadmap and the proposed design is described. The development roadmap and micro-architectures of an ALU cluster, an ALU cluster Intellectual Property and Floating point units for the ALU cluster IP and the overview of the AMBA AHB protocol are described in Chapter 3.

In Chapter 4, implementation results of proposed designs are described. The verification and testing results are also introduced in this chapter. Then the performance evaluation and comparison are discussed in the last part of the chapter. In the last chapter of this thesis, Chapter 5, the conclusion and future work are summarized.





# CHAPTER 2

## Background and Challenges

This chapter begins with the discussion of conventional programming model for media application. Therefore two kinds of programming model are used to deal with data while the comparison between programming models will be introduced in next chapter. Meanwhile, different system architectures of implementing the design are presented and discussed. These issues discussed of the programming models and system architectures are the background of current research. This thesis is motivated from these issues of the background.

### 2.1 Issues of Programming Model

Traditionally, the media applications are processed by conventional programming model implemented in conventional general purpose processor architecture. As shown in Fig 2.1, conventional programming model read data from memory system for computation and write results back into memory system. The memory system of this processing model depends on caches, which is optimized for latency and data reuse. Remind the characteristics of media processing applications. First, every stream is read exactly once, resulting in poor cache performance. Second, operating one data element is largely independent to others. It results in a large amount of data parallelism and high latency tolerance. Finally it can not support high ratio of computation to memory access. Above-mentioned issues show that large available parallelism, little data reuse and high computation to memory access ratio are cramped by the attributes of caches.

Another clincher is memory-processor communication bandwidth gap. As shown in Fig 2.2, the processor-memory performance gap reveals that the performance growth of memory is much slower than processor [8]. The phenomenon will cause more latency for memory access and communication between processor and memory

is more critical. And traditional memory system utilizes global structures to provide data bandwidth. It means that it cannot scale to multiple arithmetic logic units for high performance rates in media applications.

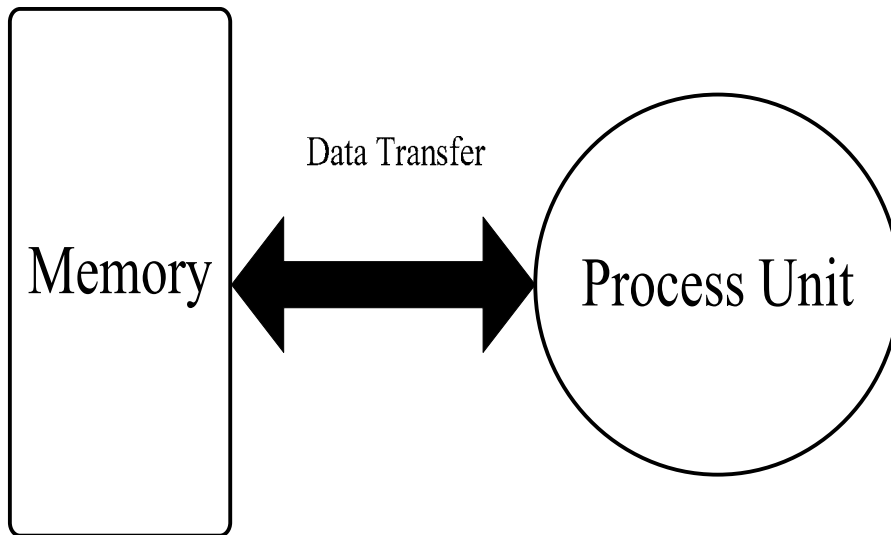


Fig 2.1 Conventional Programming Model

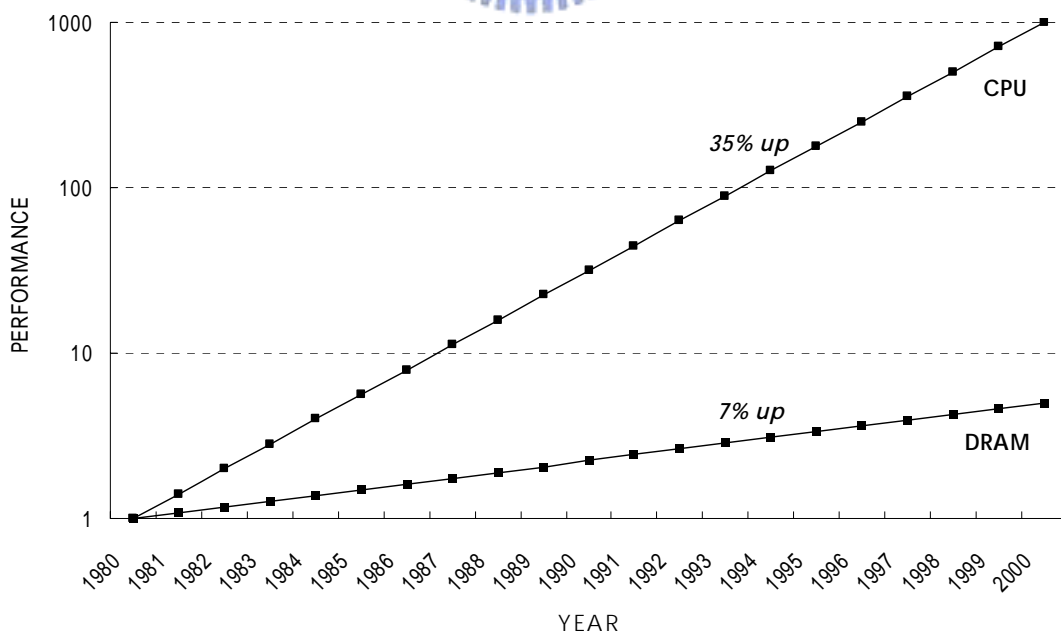


Fig 2.2 Processor-Memory Performance Gap

## 2.2 Issues of System Architecture

Generally speaking, there are many different system architectures when implementing a design. Three main system architectures of design methodology, such as application specified integrated circuit(ASIC), platform-based architecture and reconfigurable architecture will be briefly introduced in this section on the basis of time to market demands, programmability, flexibility and physical area, etc. Following, the pros and cons of these system architectures are discussed. [9 - 11]

The application specified integrated circuit is the most commonly used in these architectures. The ASIC design principle is shown in Fig 2.3. The chip implementation could be finished very quickly as long as the well-defined specification is given. Overall function and performance, such like area, power consumption and operating frequency, are optimized for the specification required. Thus long design cycles which include circuit design and the manufacture increase the investment risk. And design verifications and corrections also take a large amount of design effort. It raises investment risk also. In addition, the waste of logic resources and power dissipation for non-active hardware is another issue for the design methodology. Besides, let us consider the situation that the specifications are changed. In this situation, it reveals the lack of flexibility for ASIC design. It also shows the lack of programmability and non-reusable in the architecture.

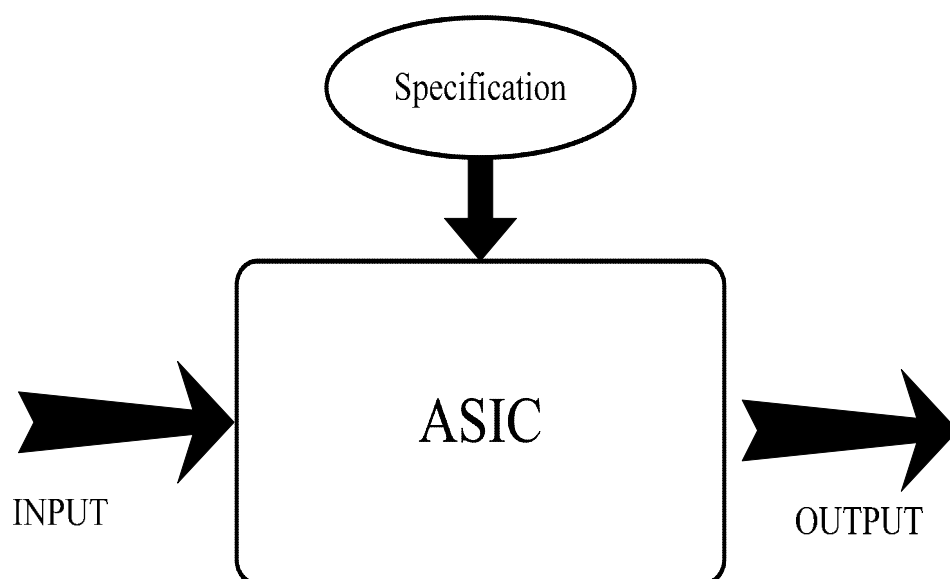


Fig 2.3 Application Specified Integrated Circuit Design

The platform-based architecture includes a processor, memory, communication bus and multiple functional hardware accelerators. It gains more flexibility than ASIC architecture from reusing existence intellectual property (IP), such as digital signal processor (DSP), baseband codec, audio applications accelerator and other functional blocks. The example of platform-based architecture is shown in Fig 2.4. Different IP blocks are added or removed to meet different application. The platform-based architecture provides a common communication bus for convenience to integrated different IP macros quickly. Different systems will be set up as fast as possible. It reduces the design and re-develops effort significantly. One more attractive thing is that these platforms have been set up with a developing baseboard. Many common IP and peripherals on the baseboard will benefit to fast prototyping. The existent OS of the baseboard can reduce the effort of connecting the real applications to development design. Current research such as [12], [13] and [14] are listed in the reference.

Thus there are some drawbacks in this architecture. The interface communicate each functional macro increases the overhead of whole system. Besides, the memory bandwidth is limited by the communication bus. These factors decrease the efficiency seriously. In the meantime, the power consumption should be increased when more IP blocks are included. The idle IP macros waste unnecessary power dissipation, too. As discussed above, the platform-based architecture is more flexible and programmable than ASIC design. Thus this architecture is still a task-oriented system. It can not be applied to any application using the same framework.

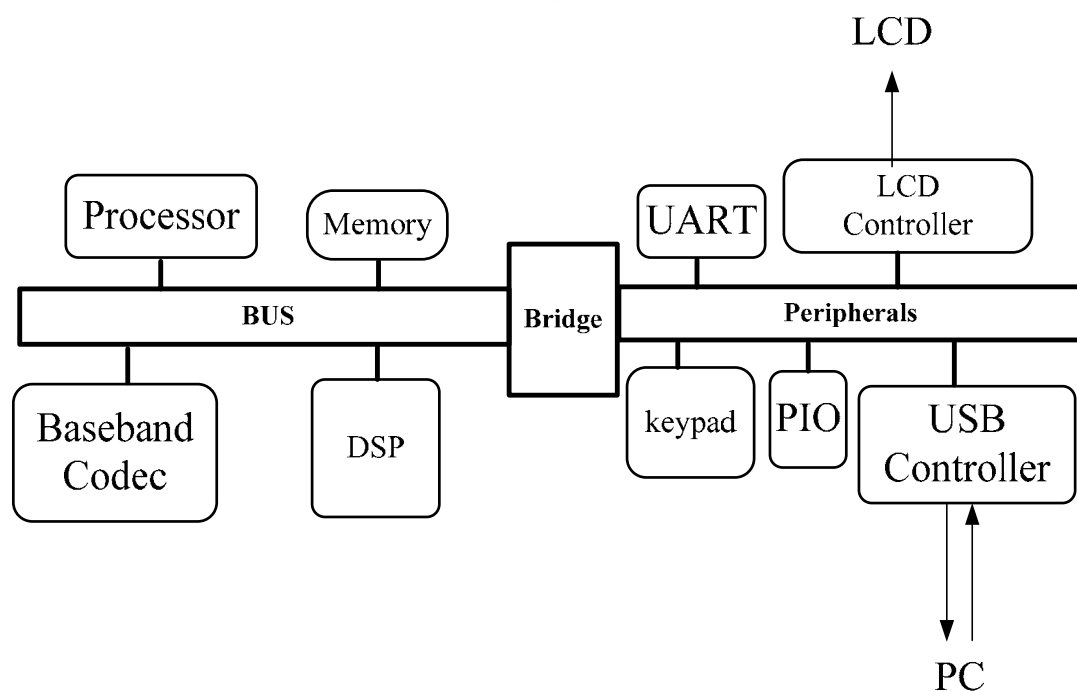


Fig 2.4 An Example of Platform-Based Architecture

The reconfigurable architecture, the third design methodology, is similar to platform-based architecture. As shown in Fig 2.5, there are multiple general processing elements in this architecture. These processing elements, or ALU cluster IP blocks, play the key role of operating data stream. A system of reconfigurable architecture is built up with a micro controller, a bus or a network on chip system and a well-hierarchical memory system. One advantage of this kind of architecture is the usages of hardware accelerator IP are reconfigurable. It provides a significant flexibility and programmability for different applications. Another advantage is the applications can be operated concurrently. It means that it provides the ability for parallel operation. Nevertheless, there exist some potential drawbacks in using the design methodology. First, without power management system the power dissipation of unused process elements can not be saved. Second, the reconfigurable architecture could not match the above-introduced characteristic very well since the bandwidth of communication bus is insufficient and data transfer bottleneck encounters between process elements and memory system. The efficient memory hierarchy system is needed to solve the performance degradation. Current research such as [15], [16] and [17] are referenced in the bibliography.

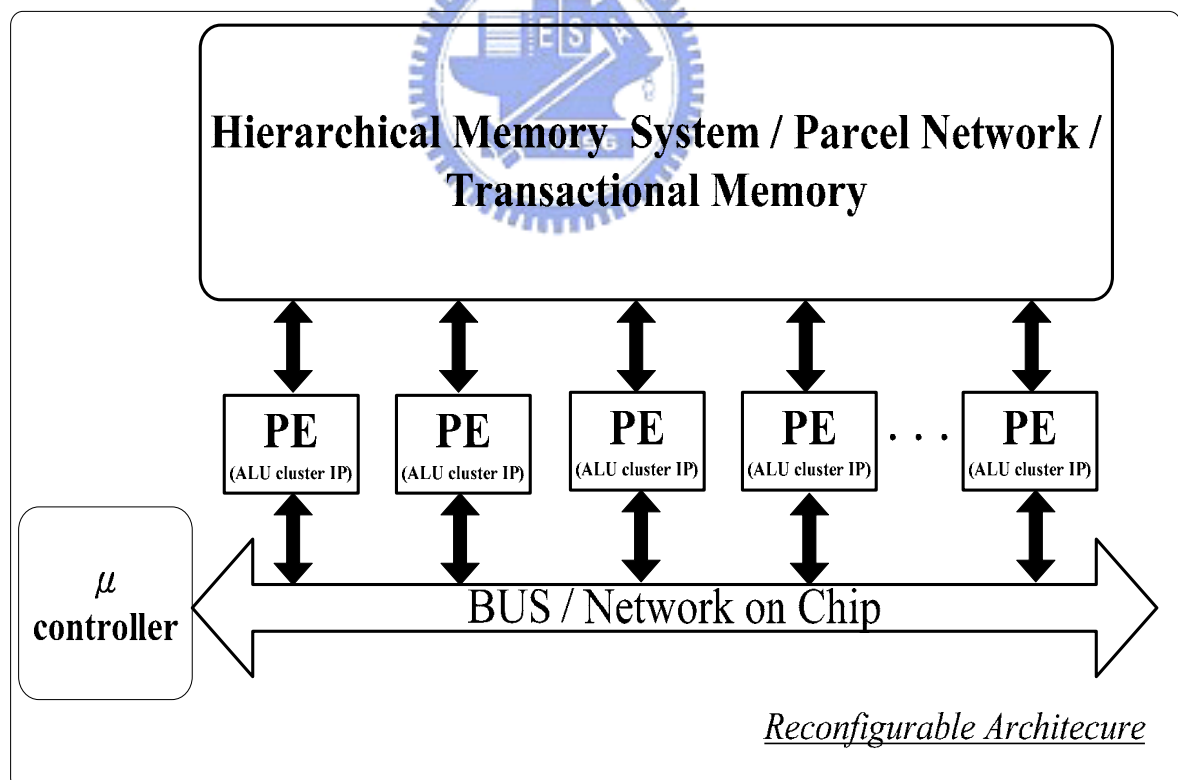


Fig 2.5 A Diagram of Reconfigurable Architecture

In conclusion, one of these system architectures can be selected to implement the design trading off between pros and cons addressed above. These pros and cons corresponding to characteristics of media applications are summarized in the Table 2.1 listed below. Thus, any one of them adopted alone suffers from some drawbacks and can not meet the application of media processing very well. Consequently, the proposed design will be addressed and discussed in later section. It must resolves these issues.

Table 2.1 System Architecture vs. media application

	ASIC	Platform-Based Architecture	Reconfigurable Architecture
System Architecture for Media Application	Lack of hardware flexibility Lack of programmability Inefficient Memory bandwidth Waste of logic resources and power to feed non-active hardware	Memory bandwidth will be limited by bus Immediate data transfer will be inefficiency Overhead of bus interface Flexibility and programmability, but task oriented	Flexibility and programmability Parallelism Data transfer bottleneck between process elements and memory system Need to solve insufficient memory bandwidth to expose locality with little global data reuse

# CHAPTER 3

## Development Roadmap and Proposed Design

Base on the previous two chapters, the background, challenges are addressed. Then the developmental roadmap and the proposed design of this thesis will discussed in this chapter.

The first section is the developmental roadmap of this thesis. It introduces the motivation to propose these designs after the issues of Chapter 2 are discussed. Afterwards the streaming programming model and developmental roadmap overcome the issues mentioned above are introduced.

The second section is the demonstration of previous design of ALU cluster. Review the architecture of the ALU cluster, the key processing element of ALU cluster intellectual property. Through testing the manufactured chip, the results confirm the correctness of functionality and the architecture is not only feasible but also efficient for media applications. The latter part of the description of this paragraph will be introduced clearly in next chapter.

The third part of this chapter is the description of the designed ALU cluster intellectual property. This design is an integration of the improved ALU cluster and the AMBA AHB slave interface. The improved ALU cluster is based on the ALU cluster discussed in previous section. The detail architectures and overview of AMBA AHB slave protocol are introduced in this section.

Eventually the forth section of this chapter is the description of the designed floating point operation units for ALU cluster IP. It will bring up an idea to integrate the floating point unit in the ALU cluster IP. The design consideration of the floating point units are described in the section. The details of the design are summarized.

## **3.1 Developmental Roadmap**

### **3.1.1 Motivation**

Issues of programming models and system architectures encountered in modern media processing system are addressed in Section 2.1 and Section 2.2. These drawbacks make the media applications handled inefficiently. The situation is more and more serious when great deals of media applications are applied in portable systems.

The media streaming architecture with homogeneous processor cores, a turnkey solution for media applications, is proposed in this thesis. This system intends to provide several cons of processing data stream. First it provides highly parallel computing ability so that multiple processing elements are needed. Performance improvement of media applications is achieved because of exploiting the large available parallelism inheritance of media process. Subsequently, the communication bandwidth bottleneck discussed above has to solve. An efficient hierarchy of memory system is needed to expose the characteristic of little global data reuse and high computation to memory access ratio in media applications. Therefore, a reconfigurable hardware accelerator is built as a processing element to form the media streaming architecture with homogeneous processor cores in this thesis.

### **3.1.2 Roadmap**

As mentioned above in the previous chapter, the conventional programming model is not suitable for these applications. So the stream programming model is adopted in this thesis. We will discuss the stream programming model below. Following the developmental roadmap including system architecture is described in this chapter later.

#### **3.1.2.1 Stream Programming Model**

In the stream programming model, data is aligned in order as a stream. Streams are arbitrary data type. Operations are applied on entire streams. These operations perform computations, stream transfers, loads and stores etc. in the programming model. Nodes which carry out these operations are called kernels. They perform computation, such as a function, to each element of whole data streams. Kernels input



one or more data streams to operate and output one or more data streams as outputs. These kernels only operate on local data and may not make arbitrary memory references.

After introducing streams, operations and kernels in the stream programming model, the structure of the model are depicted in Fig 3.1. As shown in the diagram, the stream programming model handles data by chaining operations together and makes data passing through kernels. Two example of dealing with applications using stream programming model are shown in Fig 3.2 and Fig 3.3 respectively. One is 1024-points complex radix-2 Fast Fourier Transform (FFT), a popular operation in multimedia processing [18]. Another is the example of image processing, the Stereo Depth Extraction, commonly used in modern image and medical diagnosing system [3].

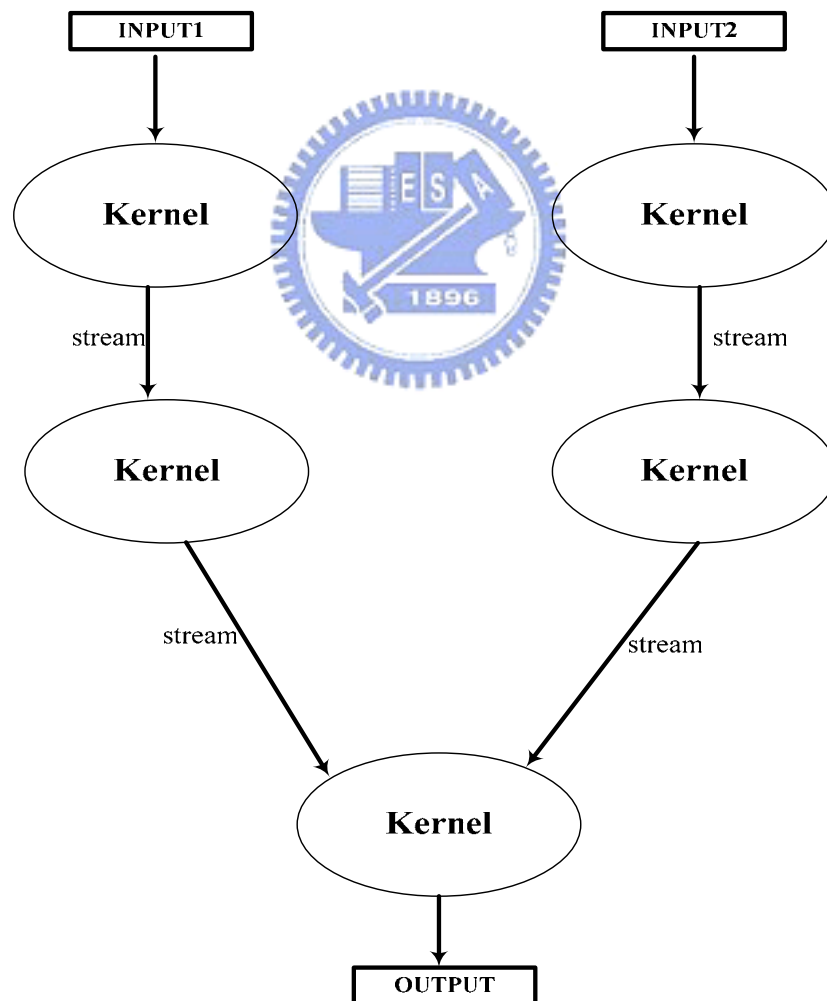


Fig 3.1 Stream Programming model

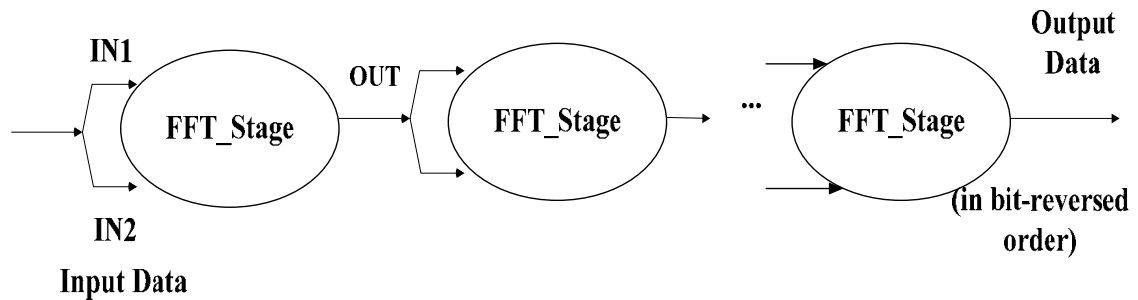


Fig 3.2 Example of 1024-points radix-2 Fast Fourier Transform

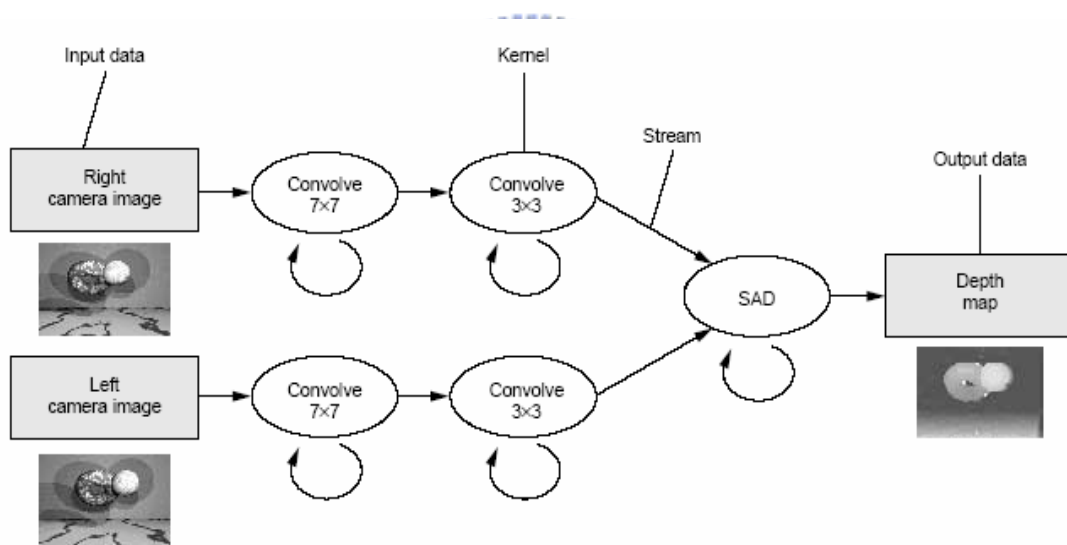


Fig 3.3 Example of Stereo Depth Extraction

Remind the characteristic of the conventional programming model discussed in Section 2.1. Comparison between the stream programming model and the conventional programming model, some obvious pros are revealed when expressing media applications in the stream programming model. Corresponding three features of media process such as little data reuse, high large available parallelism and computation to memory access ratio, Pros are described briefly below. First, multiple kernels exploit the inherent parallelism feature. Second, data streams produced at the end of one kernel will consume at the next kernel makes the programming model fit

the feature of little data reuse. Finally the high computation to memory access ratio results from the minimization global memory usage in the stream programming model. Features compare between these two programming models are listed in Table 3.1.

Table 3.1 Comparison between programming models

	Conventional Programming Model	Stream Programming Model
Large available parallelism	One central processor unit	Multiple kernels
Little data reuse	Traditional caches are ineffective	Data produced at the end of one kernel will consume at the next kernel
High computation to memory access ratio	Each element reference the off-chip memory	Minimize global memory usage

The programming model in this thesis was adopted the stream programming model. Because of the features mentioned above, it is suitable for processing system aimed at multimedia processing applications.

### 3.1.2.2 Developmental Roadmap

The developmental roadmap proposed in this section provides a suitable solution to process applications to conform the requirement expected. A sketch map of proposed development roadmap is illustrated in Fig 3.4. In this roadmap, five steps are segmented to build the media streaming architecture with homogeneous processor cores. As illustrated in the Fig 3.4, the proposed system gains the advantages from platform-based architecture and reconfigurable architecture. The mixture of system architectures are utilized as the structure of proposed system to overcome the challenge of issues deriving from using these system architectures singly. And the whole processing system provides an efficient processing system to get over issues of programming models and system architectures.

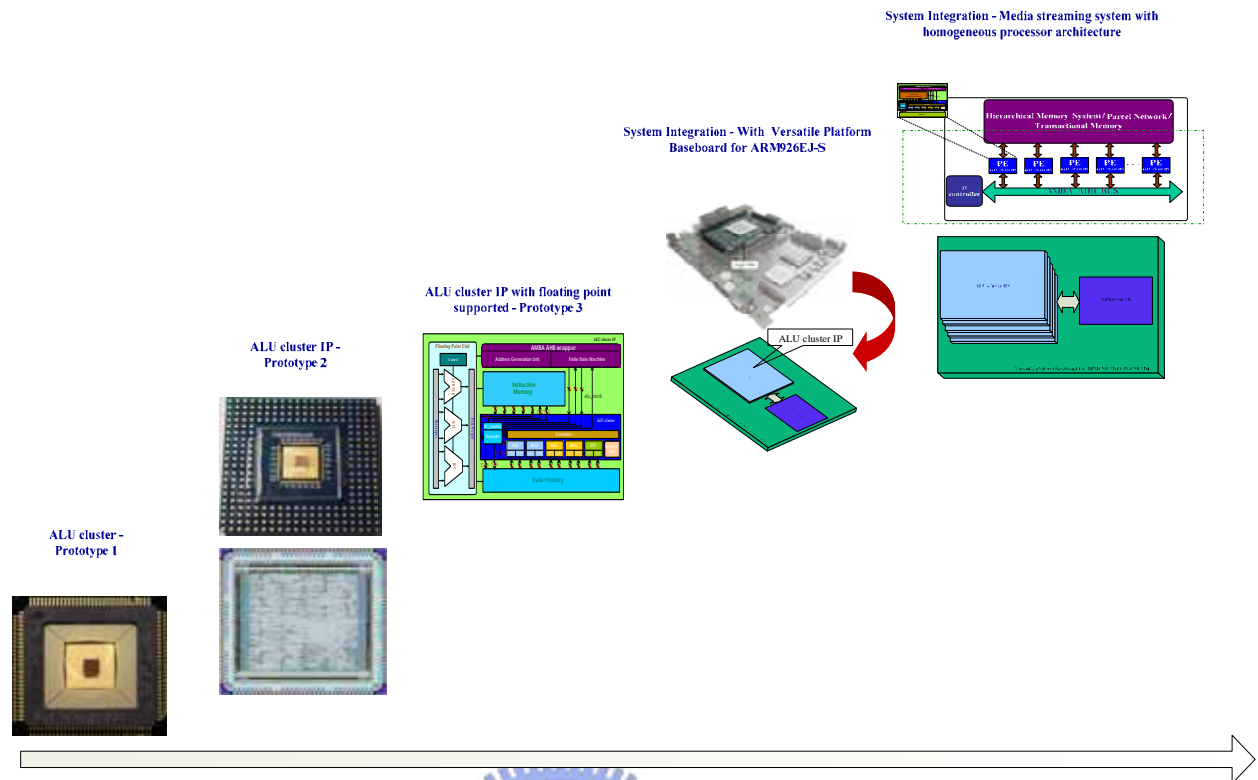


Fig 3.4 Development roadmap

Descriptions of these five steps are introduced briefly in this section. First, the leftmost chip photo, implemented in UMC 0.18um CMOS technology, in Fig 3.4 is the prototype 1 of ALU cluster [19] [20] . This is called the first step of proposed developmental roadmap. The chip had been measured and verified. As a processing element, the measurement results demonstrate that the architecture and functionality is suitable and correct for the applications.

By the right side of prototype 1, the ALU cluster IP with AMBA AHB interface, the prototype 2, was designed and taped out using TSMC 0.15um CMOS technology. The layout and die photo with package are depicted in the development roadmap respectively. This step verifies the designed ALU cluster IP to be suitable as a reconfigurable hardware accelerator in media applications. In addition to this purpose, the interface obeyed the AHB slave bus provides a common communication bridge between these ALU cluster IPs and micro-controller used to manage whole media streaming system. These features make the proposed prototype 2 have ability as a processing element to be applied in the media streaming system. It is a significant feature of the second step in developmental roadmap.

In modern multimedia applications, the floating point operations occupy a large percentage of the computation amount. These floating point operations stand a key role of performance and power consumption in whole applications. A floating point unit is essential in performance improvement if the budget of power consumption and logic resources are agreed. Consequently, an ALU cluster IP supported floating point operation is a requirement. As shown in the middle of Fig 3.4, an ALU cluster IP with floating point supported are introduced. The hard macro of floating point unit is designed and implemented using TSMC 0.18um CMS technology. The combinations of ALU cluster IP and the hard macro provides efficient computing ability to handle floating point operations. It is the third generation in the described developmental roadmap.

Following the forth step in Fig 3.4 is discussed in this paragraph. In this step, system integration is preliminary started. An ALU cluster IP with compatible board for logic tile connector will be stacked into the RealView versatile platform baseboard for ARM926EJ-S [21] [22]. In this baseboard, the ALU cluster IP is verified whether the AHB bus interface fits the bus protocol. One ALU cluster IP stacked in the board are also made sure that the IP has the ability as a hardware accelerator integrated with the platform.

Finally multiple processing elements, ALU cluster IPs, will be combined with the versatile baseboard to form a media streaming architecture with homogeneous processor cores. The proposed system matches the features of media applications such as large available parallelism, little data reuse and high computation to memory access ratio. Suitable benchmarks are ported into this processing system and compare with each other. These applications include some popular operations in multimedia applications and MIMO-OFDM system. The finite impulse response (FIR) filter system and Fast Fourier Transform (FFT) are selected as benchmarks in media applications. And the key operations of MIMO-OFDM system, such as matrix inversion and Gram-Schmitt process, are selected as benchmarks, too [23]. Adoption of stream programming model and mixture of system architectures make the media streaming system become a turkey solution for modern multimedia processing requirement.

Five paragraphs discussed above are introduced the proposed developmental roadmap compendiously. The details of these steps, such as micro architectures, implementation results and etc., are discussed and described in the following sections and chapters respectively.

## 3.2 An ALU cluster

The briefly description of the previous design, an ALU cluster, included the micro architecture is described. The ALU cluster is called prototype 1 in the above-presented developmental roadmap. It is convincible that it can handle media applications expectedly.

### 3.2.1 Micro-Architecture of an ALU cluster

As the major part for handling the media processing, an ALU cluster includes five arithmetic units, supporting to process the parallel data concurrently. As shown in Fig 3.5, they are two ALUs, two multipliers and one divider. Large amount of digital signal processing application are suitable for porting in the architecture with mixture of arithmetic units. There is also one scratch pad register file (SPRF), ten banks of intra register file (IRF), a controller and a decoder.

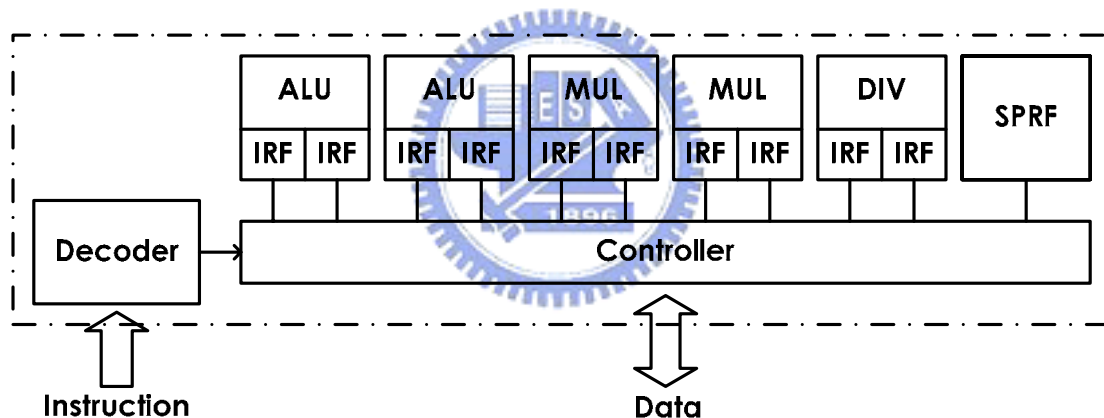


Fig 3.5 Micro-Architecture of ALU cluster

There are thirteen instructions can be executed by the ALU, such as ADD, SUB, ABS, AND, OR, XOR, NOT, SLL, SRL, SRA, LT, GT, EQ. The adder and comparator adopted for ALUs are the carry-lookahead architecture with two stage pipeline. Booth encoding architecture was adopted for our four stage pipeline multiplier. This multiplier can carry out multiplication. The last arithmetic unit is the divider. It performs the division operation that gets the quotient and remainder and calculates the square root. The designed divider in an ALU cluster is not the key kernel about performance concerned so that this unit is not pipelined and considered to shrink the logic resource by increasing latencies of operation.

As mentioned above, in the ALU cluster there are IRFs embedded for each operation units. These intra register files are local to themselves arithmetic units. The purpose of IRFs is to provide an efficient memory bandwidth for arithmetic units. In other words, these arithmetic units would not waste the precious global memory bandwidth. It acquires required bandwidth from less precious bandwidth of IRFs. Another extra storage element inside the ALU cluster is scratch pad register file. The capabilities of SPRF are as the storage element for commonly used coefficient of applications.

The remaining parts of the ALU cluster are the decoder and the controller. The decoder can decode the instructions from off-chip instruction memory and provides control signals needed for ALU cluster. During the execution of an ALU cluster, the controller sequences and issues the decoded instructions to the function units and decides the inputs source, such as IRF, SPRF or data memory. When an ALU cluster operates to read or write, the controller manages the data flowing.

### **3.3 An ALU cluster Intellectual Property**

In this section, an ALU cluster Intellectual Property (IP) is designed and the architecture is also discussed. It is the prototype 2 mentioned in the roadmap presented in previous section [24]. As mentioned in previous chapter, there is an AMBA AHB wrapper in the ALU cluster IP. First the AMBA AHB protocol will be described in the following paragraph. Then the micro architecture of an ALU cluster IP will be presented and discussed in the last part of this section.

#### **3.3.1 Overview of AMBA**

The Advanced Microcontroller Bus Architecture (AMBA) specification defines an on-chip communications standard for designing high-performance embedded microcontrollers [25]. There are three buses defined in this specification. They are Advanced High-performance Bus (AHB), Advanced System Bus (ASB), and Advanced Peripheral Bus (APB). These bus protocols are used in different applications. For example, the AHB are used as the high-performance system backbone bus. It is for high-performance and high clock frequency system modules. It provides the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions. The ASB is also for high-performance system modules. It is suitable for system bus that the high-performance features of AHB are not required. It also supports the efficient

connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions the same as AHB. Finally, the APB optimized with minimal power consumption and interface complexity is used for peripherals. APB can be used in conjunction with either version of the system bus.

Four key requirements are satisfied by AMBA specification. They are right-first-time, technology-independent, modular system design and minimization of the silicon infrastructure. The system obeyed AMBA protocol could facilitate the right-first-time development of embedded microcontroller products with one or more CPUs or signal processors. The specification is technology-independent and ensures that highly reusable peripheral and system macrocells can be migrated across a diverse range of IC processes and be appropriate for full-custom, standard cell and gate array technologies. It also improves processor independence, providing a development roadmap for advanced cached CPU cores and the development of peripheral libraries to encourage modular system design. The system using AMBA protocol can be minimized the silicon infrastructure required to support efficient on-chip and off-chip communication for both operation and manufacturing test.

A typical AMBA-based microcontroller is composed of a high-performance system backbone bus (AMBA AHB or AMBA ASB) which is able to sustain the external memory bandwidth, on which the CPU, on-chip memory and other Direct Memory Access (DMA) device reside. The diagram of AMBA-based system is shown in Fig 3.6. The backbone bus of whole system has ability to provide a high-bandwidth interface between elements involved in the majority of transfers. APB, a lower bandwidth bus, is located on the high-performance bus by the bridge. Most of the peripheral devices in the AMBA-based system are located.

The features of AHB, ASB and APB are listed in Fig 3.6. As shown below in this figure, AHB is suitable for high performance, pipelined operation, multiple bus masters, and burst transformation and split transactions. Compare with AHB, ASB is lack of the ability to burst transformation and split transactions. The simple interface is adopted by APB. It latches address and control signal to save power. Thus it is suitable for many peripherals. The difference between APB and AHB or ASB is that AHB and ASB are able to wait the transfer during it is not ready whether the wait situation is from on-chip bus or itself. APB must response the transaction immediately.



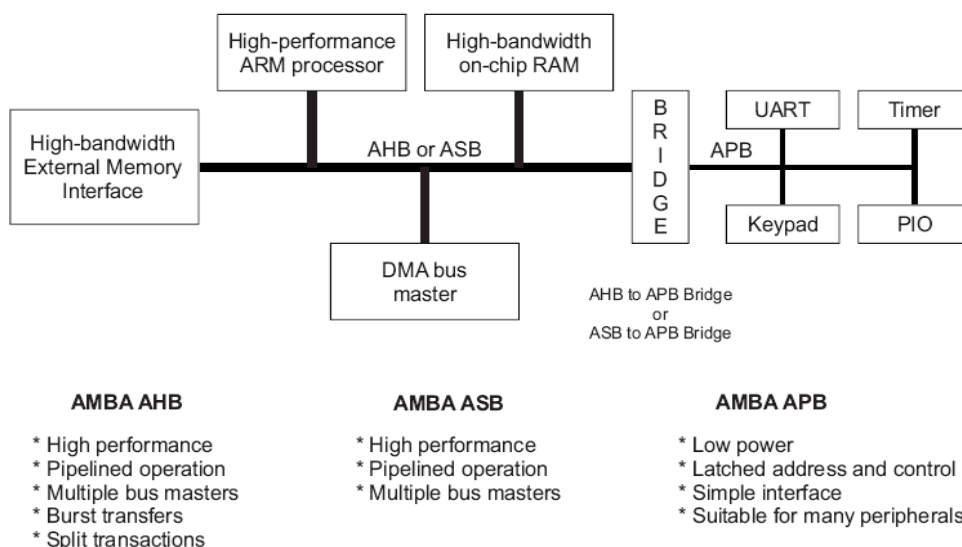


Fig 3.6 Diagram of AMBA-based system

### 3.3.1.1 Introduction of AMBA AHB

The AMBA AHB is a new generation bus intended to address the requirements of high-performance synthesizable designs. It sits above the APB and implements the features required for high-performance, high clock frequency system, including split transactions, burst transfers, single cycle bus master handover, single clock edge operation, non-tristate implementation and wider data bus configurations. The AMBA AHB system is designed with the following components, including AHB bus master, slave, decoder and arbiter. These typical components are described briefly below.

The AHB master starts an AMBA AHB transfer by driving address and control signals. They provide information which the AHB slave needed. Exact one bus master is allowed to actively use the bus at any one time. This component is the most complex bus interface in an AMBA system. Usually the designer would use existence bus master rather than concerned with the details in the bus master interface.

The AHB bus slave responds a write or read operation. All signals required for the transfer, such as the address and control information, will be generated by the bus master. The bus slave signals back to the active master the success, failure or waiting of the data transfer.

The AHB bus decoder is used to perform a centralized address decoding function, which improves the portability of peripherals, by making them independent of the

system memory map. It is used to decode the address of each transfer and provide a select signal for the slave that is involved in the transfer.

Finally the AHB bus arbiter will be described. The AHB arbiter is used to control which master has access to the bus. It ensures that only one bus master is allowed to initiate data transfers at a time. The arbiter uses a prioritization scheme to decide which bus master is currently the highest priority master requesting the bus. The detail of the priority scheme is not specified and is defined for each application.

Some details of AHB interface, such as bus interconnection, signals for AHB slaves, basic transfer, transfer type, address decoding and burst operation will be presented in the following sub sections. The remaining details will be described in the AMBA specification.

### **3.3.1.2 Bus Interconnection**

The diagram of AMBA AHB interconnection is shown in Fig 3.7. As illustrated in the diagram, this protocol is designed to be used with a central multiplexer interconnection scheme. The scheme makes all bus master drive out the address and control signals indicating the transfer they intend to perform. Then the arbiter obeyed the prior policy determines which master is able to route its address and control signals to all of the slaves. There is also a central decoder in Fig 3.7. It is used to control the read data and response signal multiplexer to response appropriate signals from the slave to the master which involved in the transfer.

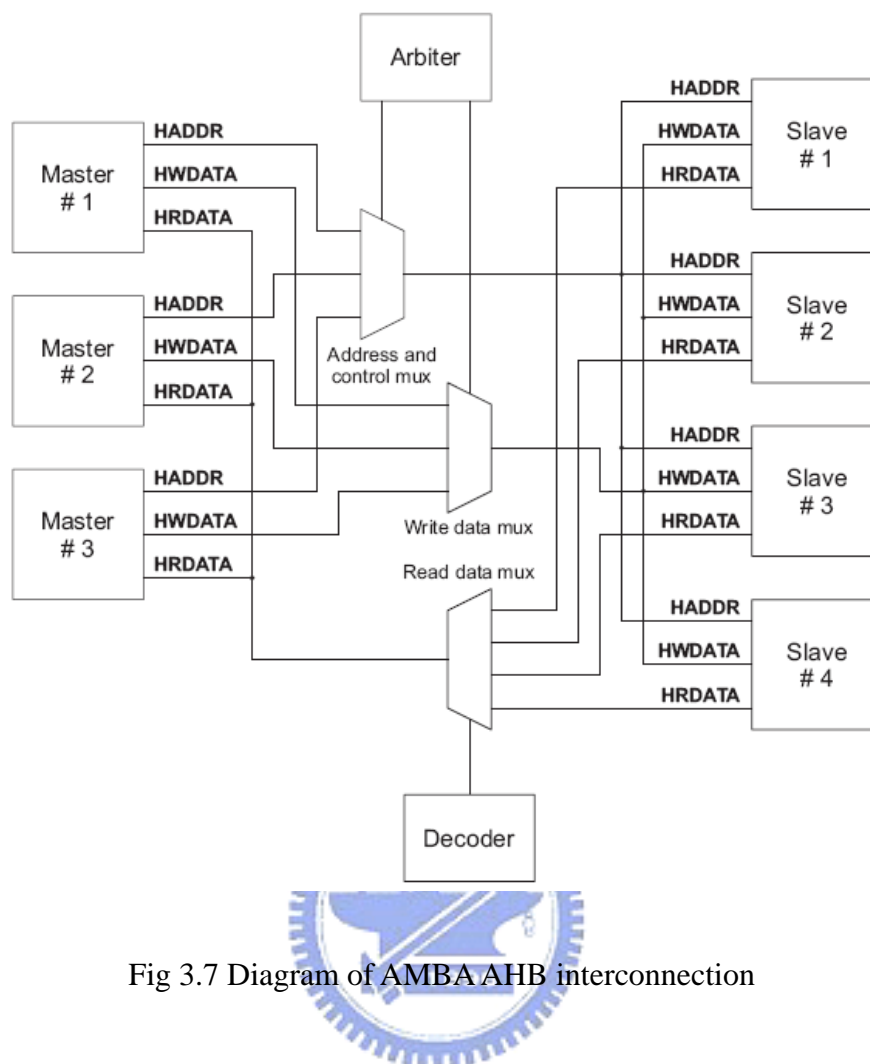


Fig 3.7 Diagram of AMBA AHB interconnection

### 3.3.1.3 Signals for the protocol of AMBA AHB slave

The signals using in AHB slave protocol are shown in Fig 3.8, which is the diagram of AHB slave interface. These signals are briefly described in the following paragraph.

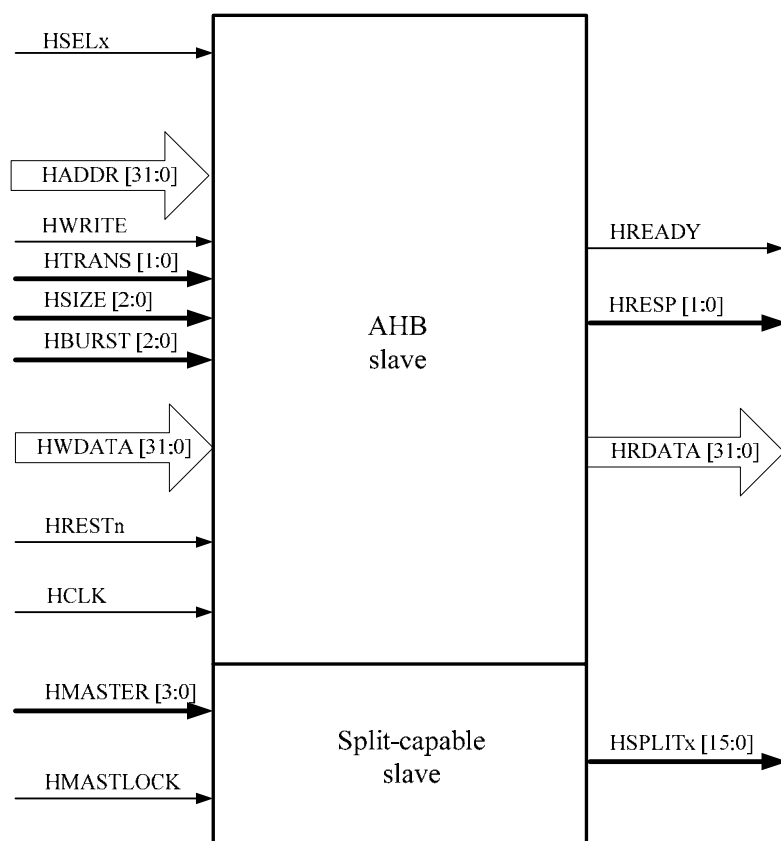


Fig 3.8 Diagram of AHB slave interface

The input signals are classified into six groups such as select, address and control, data, reset, clock and split capable signals. The select signal, HSELx, is from decoder and indicates that the current transfer is intended for the selected slave. The address and control signals such as HADDR, HWRITE, HTRANS, HSIZE and HBURST are from master to slave. HADDR is the system address bus. HWRITE indicates that the transfer is reading operation or writing operation. HTRANS indicates the type of current transfer. HSIZE shows the size of the transfer and HBURST shows the burst type of operations in the transfer. The data signal, HWDATA, is form master and used to transfer data during write operations. The reset and clock signal are HRESETn and HCLK respectively. The signals shown in the bottom of Fig 3.8 are from arbiter and used to support the split transactions.

There are four output signals. They are HREADY, HRESP, HRDATA and HSPLITx. The HREADY indicates a transfer has finished on the bus and the HRESP provides additional information on the status of a transfer. Above-mentioned signals

are called transfer response signal group. The HRDATA is used to transfer data from slaves to masters during read operations. The last output signal is HSPLITx, which is used for split completion request.

### 3.3.1.4 Basic Transfer

The basic transfer of AHB protocol is composed of two phases. They are address phase and data phase. An example of simple transfer is shown in Fig 3.9. As illustrate in the figure, the address phase only requires one cycle but the data phase may require several cycles. The necessary signals needed for the basic transfer are HCLK, HADDR, control, HWDATA. The transfers will response the HREADY, HRDATA and HRESP. The figure demonstrates how the address and data phases of the transfer during different clock periods. The address phase always occurs during the data phase of previous transfer. The above-mentioned situation of overlapping is based on the pipelined nature of the bus and allows for high performance operations. The logic high of HREADY represents the transfer is ready to be finished and logic low shows that the transfer is needed to be extended. The example of the transfer needed to be extended is shown in Fig 3.10. The address phase is the same as Fig 3.9. Thus the data phase shown in Fig 3.10 is extended with two cycles because the transfer is not ready to be completed by means of signaling the HREADY logic low. This may result from both master and slave depends on the transfer type. It will be introduced in the following section.

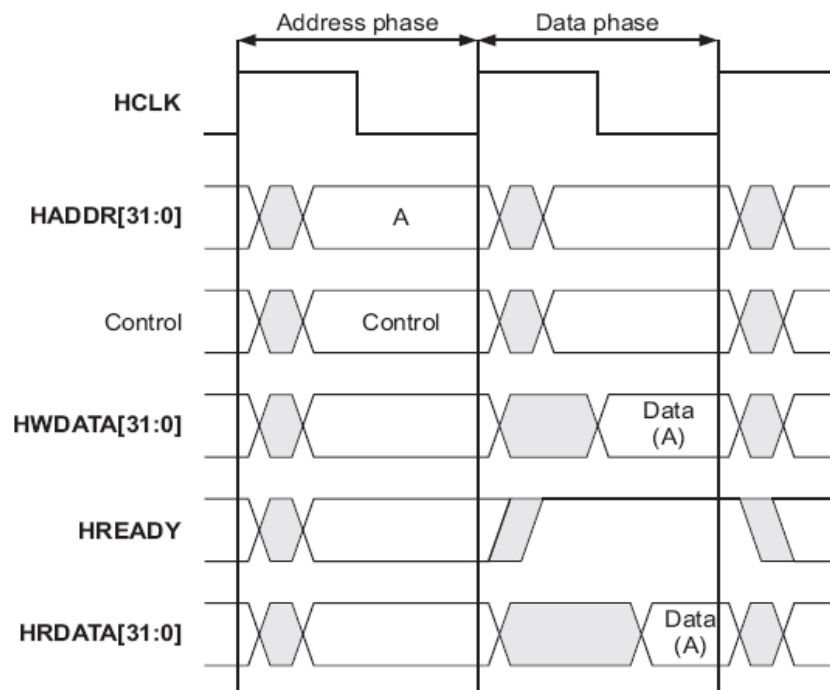


Fig 3.9 An example of simple transfer

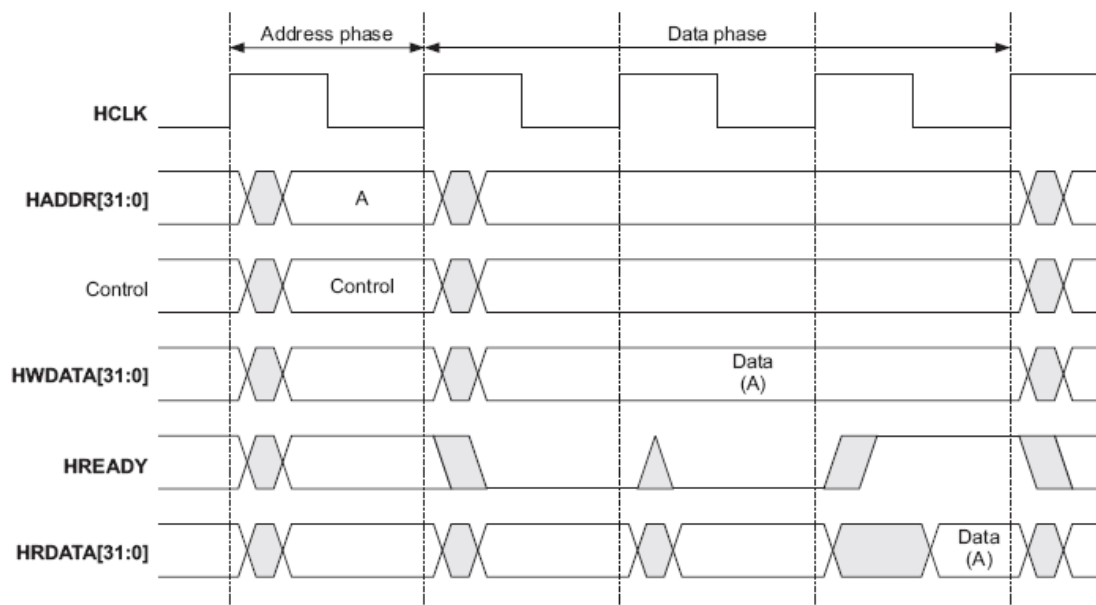


Fig 3.10 The example of the transfer extended

### 3.3.1.5 Transfer Type

The transfers of AMBA AHB are classified into four different transfer types. The HTRANS signals will be used to indicate the type of transfer. The two bits signal represents IDLE, BUSY, NONSEQ and SEQ by 00, 01, 10 and 11 respectively. These transfer types will be introduced below.

The IDLE state indicates no data transfer is required and is used when a bus master is granted the bus but does not intend to perform a data transfer. The bus slaves must always provide a zero wait state OKAY response to IDLE transfers and the transfer should be ignored by the slave.

The BUSY transfer type indicates the bus master is continuing with a burst of transfers, but the next transfer cannot take place immediately. This transfer type allows bus masters to insert IDLE cycles in the middle of bursts of transfer. When a master uses the BUSY transfer type the address and control signals must reflect the next transfer in the burst. The transfer should be ignored by the slave. Slaves must always provide a zero wait state OKAY response, in the same way that they respond to IDLE transfers.

The NONSEQ transfer type is used to indicate the first transfer of a burst or a single transfer. The necessary information such as address and control signals are

unrelated to the previous transfer. In AMBA AHB specification, a single transfer on the bus is treated as the first one of a burst therefore the transfer type is NONSEQ too.

The last one transfer type is SEQ. This type indicates the remaining transfers in a burst. The address needed is related to the previous transfer and it is equal to the address of the previous transfer plus the size in the incrementing burst. But in the situation of wrapping burst, the address of the transfer wraps at the address boundary equal to the size multiplied by the number of beats in the transfer. In addition, the control information is identical to the previous transfer.

### 3.3.1.6 Address Decoding

The select signal, HSEL<sub>x</sub>, will be provided by an address decoder shown in Fig 3.11 for each slave on the bus. The select signal is a combinatorial decode of the high-order address signals and simple address decoding schemes are encouraged to avoid complex decode logic and to be suitable for high-performance operations. A slave only samples the address and control signals and HSEL<sub>x</sub> when HREADY is logic HIGH. It indicates that the current transfer is completing. Under certain situation it is possible that HSEL<sub>x</sub> will be asserted when HREADY is logic LOW, but the selected slave will have changed by the time the current transfer completes. The minimum address space can be allocated to a single slave is 1kB. All bus masters are designed such that they will not perform incrementing transfer over a 1kB boundary, thus ensuring that a burst never crosses an address decode boundary.

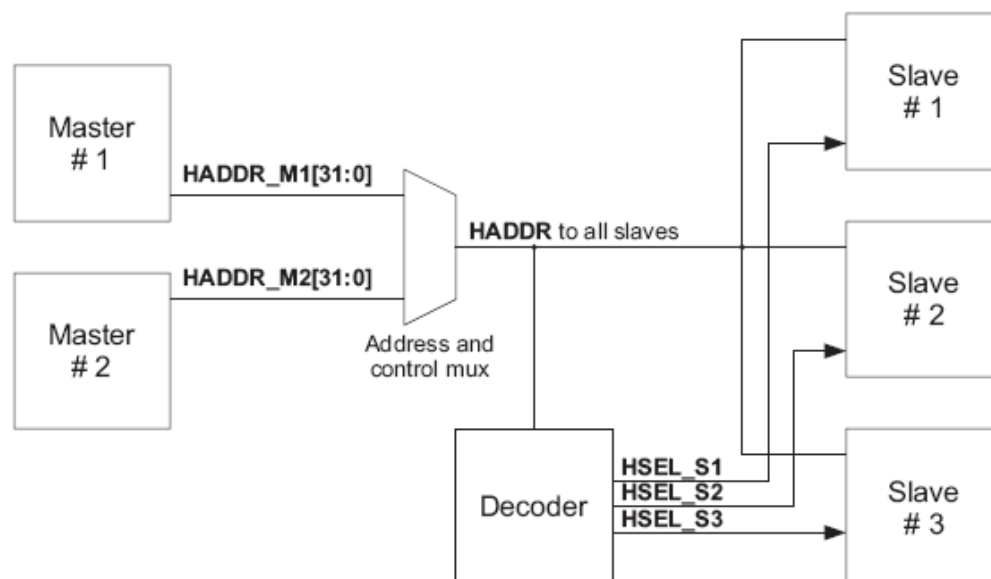


Fig 3.11 Slave Selected Signal

### 3.3.1.7 Burst Operation

There are two kinds of bursts of the burst operations supported in the AMBA AHB protocol. They are incrementing and wrapping burst modes. Four, eight and sixteen-beat bursts are defined in AHB protocol. The burst information is identified by the signal HBURST. It decides the burst modes and beat number. The relationship between signal and type is listed in Table 3.2. There are eight types defined in this table.

Table 3.2 Burst Signal Encoding

HBURST [2:0]	Type	Description
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP4	4-beat wrapping burst
011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	16-beat wrapping burst
111	INCR16	16-beat incrementing burst

The address accessing of each transfer in the burst of the incrementing burst mode is sequential and an increment of the previous address. In the wrapping burst mode, if the start address of the transfer is not aligned to the total number of bytes in the burst (size x beats) then the address of the transfer in the burst will wrap when the boundary is reached. For example, a four-beat wrapping burst of word accesses will wrap at 16-byte boundaries. Therefore, if the start address of the transfer is 0x34, then it consists of four transfers to addresses 0x34, 0x38, 0x3C and 0x30. It will wrap the address back when the boundary is reached. As description it will wrap back to 0x30.

Bursts must not cross a 1kB address boundary. It is important that masters do not attempt to start a fixed-length incrementing burst which would cause this boundary to be crossed. It means that an incrementing burst can be of any length, but the upper limit is set by the fact that the address must not cross a 1kB boundary. The signal, HSIZE, is used to control the transfer size. It supports eight different sizes such as 8, 16, 32, 64, 128, 256, 512 and 1024 bits. Finally the endian policy defined in this protocol is shown in Table 3.3 and Table 3.4. They are big-endian and little-endian



respectively. The designer only needs to obey one of the endian policies and makes whole systems consistently.

Table 3.3 Active Byte Lanes for a 32 bits big endian data bus

Transfer size	Address offset	Data [31:24]	Data [23:16]	Data [15:8]	Data [7:0]
Word	0	✓	✓	✓	✓
Halfword	0	✓	✓	-	-
Halfword	2	-	-	✓	✓
Byte	0	✓	-	-	-
Byte	1	-	✓	-	-
Byte	2	-	-	✓	-
Byte	3	-	-	-	✓

Table 3.4 Active Byte Lanes for a 32 bits little endian data bus

Transfer size	Address offset	Data [31:24]	Data [23:16]	Data [15:8]	Data [7:0]
Word	0	✓	✓	✓	✓
Halfword	0	-	-	✓	✓
Halfword	2	✓	✓	-	-
Byte	0	-	-	-	✓
Byte	1	-	✓	✓	-
Byte	2	-	✓	-	-
Byte	3	✓	-	-	-

### 3.3.2 Micro-Architecture of an ALU cluster Intellectual Property

The proposed ALU cluster Intellectual Property (IP) is described in this section. The detail architecture is shown in Fig 3.12. As illustrated in Fig 3.12, four main blocks composed of this design are AMBA AHB wrapper, ALU cluster, instruction and data memory. The instruction and data memory are used to feed the data and instruction required for operation into functional units.

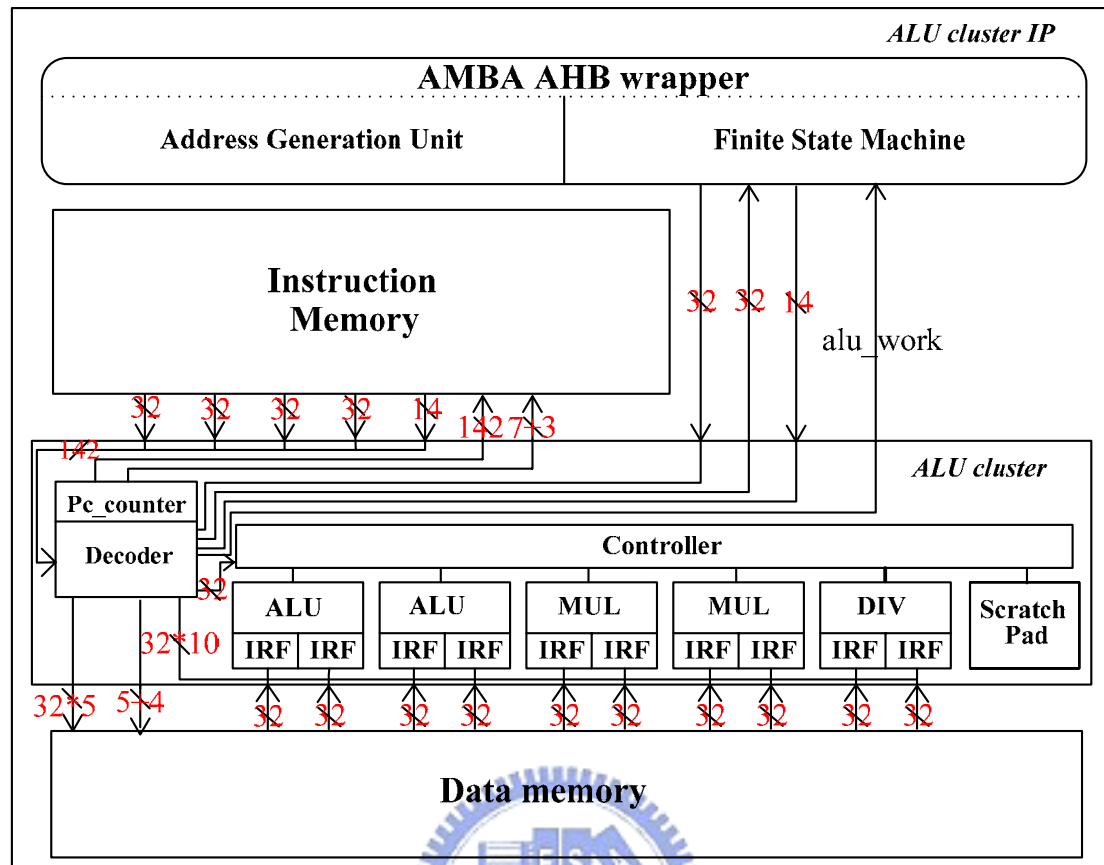


Fig 3.12 The Proposed ALU Cluster IP Architecture

The major part for dealing the media applications is an ALU cluster as description in Section 3.2. The arithmetic units and internal storages part of the ALU cluster in this ALU cluster IP is the same as the one introduced in Section 3.2.1. However, the control and internal storages are improved in this designed ALU cluster IP. The ALU cluster in this designed is improved the ability to reading source and writing destination. It makes all banks of data and instruction memory expose to the AMBA bus. It means that these memory banks can be accessed directly from AMBA bus through the AMBA AHB wrapper which will be introduced later. In addition, the better performance is exploited by shortening reading cycles. In the original ALU cluster, the reading must take four cycles to access one burst reading operation. However, in the improved ALU cluster, the reading operation takes two cycle latencies in burst reading and then the data is read sequentially in every cycle.

The ALU cluster IP must has the ability to execute when the AMBA bus is granted by other masters so that the ALU cluster needs a functional block to feed address to the instruction memory automatically. As illustrates in Fig 3.12, the Pc\_counter is used to process this job. It will increase the program counter by one in

every clock cycle. The decoder will compare the value of program counter with the end value of Pc\_counter every cycle to check if the ALU cluster finishes the job. If the job is completed, the alu\_work signal is activated to send information to the wrapper. In the alu\_work signal is inactive, the IP can not be accessed and returns RETRY signal response to AMBA bus. Besides, one special input signal combination can clear the end value of Pc\_counter in the decoder and force the IP to stop execution. The special mechanism is designed in order to avoiding the possibility of the deadlock occurrence.

Another key component of ALU cluster IP is AMBA AHB wrapper. It will be discussed in this paragraph. The wrapper interface conforms to Advanced Microcontroller Bus Architecture (AMBA) Advanced High-performance Bus (AHB) protocol described in Section 3.3.1. It provides a common interface to integrate the proposed design with ARM versatile baseboard and form a media processing system. A finite state machine (FSM) and an address generation unit (AGU) are composed of the architecture of proposed wrapper. The finite state machine of proposed wrapper is used to control the states and response the request of AMBA bus. It provides the communication capability between AHB slave bus and the ALU cluster inside proposed IP. It receives signals from AMBA bus and activates the ALU cluster to response. The FSM also controls the address generation unit to produce necessary address for the ALU cluster, whether operating in incrementing mode or wrapping mode of burst operation.

This FSM is designed with six states. They are Idle, Accessible, ALU\_Work, Un-readable Wait, Un-writable Wait and Error. As shown in Fig 3.13, the state diagram of the finite state machine, the FSM will stay in the Idle state while the IP is not accessible or the operation of ALU cluster is finished. Whether IP has done the work or suffers from some error, it returns back to the Idle state. In this state, the wrapper will be ready to receive signals from bus and prepare next operation. It will go to other starts while the bus is granted and the IP will be accessed or the ALU cluster is activated. The condition of going to other state is only when the HTRANS signal equals to NONSEQ. If the NONSEQ is encountered, it identifies which operation of the IP is requested by HWRITE then the FSM will move to the target state.

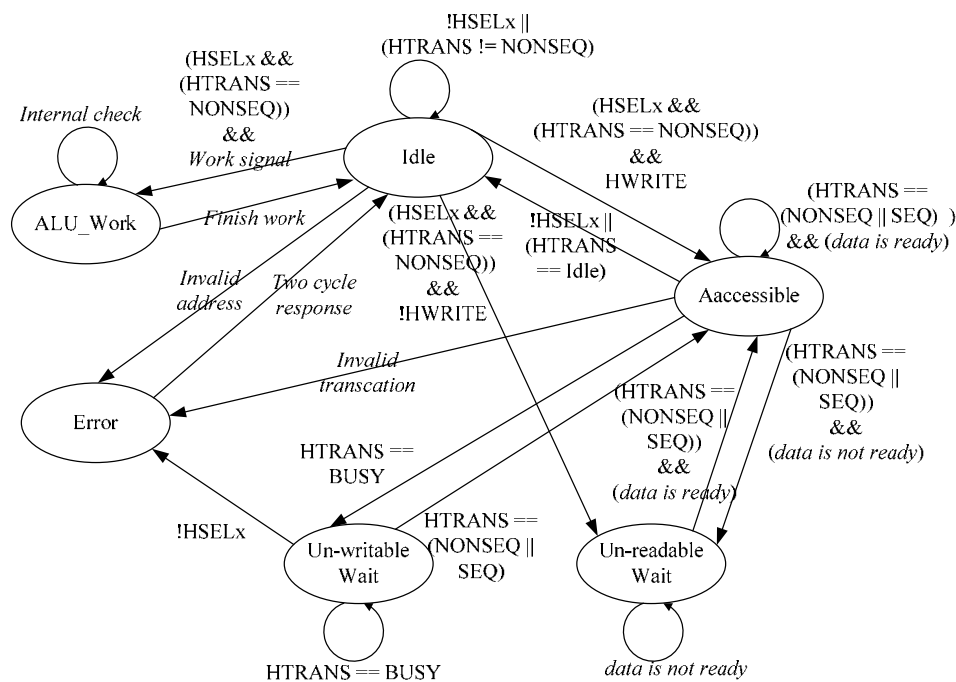


Fig 3.13 The state diagram of the finite state machine

In the next state, Accessible state, the IP is accessible. When the HTRANS signal is equal to NONSEQ and the HWRITE signal is logic high, it will directly move to this state. There is a control signal to identify the different types of accessing whether incrementing mode or wrapping mode is utilized in the burst transformation while staying this state. One type is that the IP is accessed with different address with the HTRANS signal equals to NONSEQ. Another one is that the IP is accessed continuously with the address of the previous access in wrapping or incrementing mode in the burst transformation. Three conditions are forced the FSM to other states. These cases are access is finished, ready to read but data is not ready and busy to write. The states are moved to Idle, Un-readable Wait and Un-writable Wait. The later two of the above-mentioned states are addressed below.

The Un-readable Wait state exists because of the two necessary cycle of reading data latency. One of two paths makes the FSM enter the Un-readable state is when the FSM is in the Idle state and the HTRANS signal is NONSEQ and the HWRITE signal is logic low. It presents the IP is being read. The first reading operation needs two cycles to prepare necessary data so it must be in this state until the data is ready. Then it will enter the Accessible state to perform the following reading request. Another one of two paths is from Accessible state to Un-readable Wait state because of the necessary latencies. In addition, when the IP is being written data in burst mode of wrapping or incrementing type thus the TRANS signal of AHB slave is changed to

BUSY, the FSM will enter the Un-Writable Wait state. After the signal of TRANS release from BUSY to NONSEQ or SEQ, the FSM will return from the Un-writable state to Accessible state.

The last two states of design six-stated FSM are Error and ALU\_Work state. When proposed IP is accessed illegally due to invalid address and transaction, the finite state machine will go to Error state. The invalid address and transaction result from the depth limitation of data and instruction memory. The other reason entering this state is that the IP is being accessed but is not granted expectedly. When these two cases happened, Error state will be entered and escapes from violating AMBA AHB protocol. If the Error occurs, the Error state must obey the AHB protocol and thus have two cycles response to reply the bus with proper HREADY and HRESP signal as defined in the AMBA AHB specification.

Finally, The ALU\_Work state reveals that the applications are being processed in ALU cluster. From Idle state is an only one path into the state. Whether accessed by reading or writing operations, the FSM has the ability to transfer a two cycle response to the AHB bus in the ALU\_Work state. Additionally the ALU cluster keeps working without being affected by any unexpected access until finishing the operations. Eventually there is one characteristics related to the wrapper. That is data and instruction memory embedded in the IP can be access directly by proposed wrapper.

As description of the ALU cluster IP, there is one thing needed to be reminded. One instruction must be completed through many stages so it takes more cycles to write the executed results back. The ALU is a two stage pipelined structure unit so that it takes six cycles, including two extra cycles and four necessary cycles for every operation such as instruction decoding, data source selection and results writing. Then the four stages pipelined multiplier will need eight cycles and the divider will need twenty cycles to write back the results.

### **3.4 Floating Point Units for the ALU cluster IP**

In modern media application system, floating point operation is indispensable for any applications. The floating point operation units (FPUs) are designed and implemented to be integrated with original ALU cluster IP. In the following, the design considerations of the floating point units are described. Then the implementation results of them are discussed in latter chapter. Besides, the performance evaluation of the selected benchmark is compared between the

architecture of ALU cluster IP and the architecture of ALU cluster IP supported by the floating point units in next chapter too.

### 3.4.1 Design Consideration

The floating point operation units are designed for the ALU cluster IP in order to make it more suitable and widely applied for media processing applications. Consider the architecture of ALU cluster IP, it is not well-matched for the floating point operation using the IEEE 754 standard format for floating point arithmetic [26]. In the architecture of original ALU cluster IP, the floating point operations obeyed the IEEE754 format need to be decomposed into several fields to finish the calculations and the field is easy to encounter the mistake results from the saturation problems. Consequently, the floating points units are designed for the IEEE 754 standard single precision floating point format.

The briefly review of IEEE 754 standard for binary floating point format is introduced in the following. The format of floating point numbers includes four types which are identified by its precision. They are single precision, double precision, single extended precision and double extended precision floating point number formats. The numbers of bits used to represent the value are 32 bits, 64 bits, larger or equal to 43 bits and larger or equal to 79 bits respectively. The last two formats, single extended precision and double extended precision, are not commonly used. The features of single precision format and double precision format will be focused.

As illustrated in Table 3.5, the single precision format and double precision format are listed. As shown in the table, the IEEE 754 floating point numbers have three basic fields such as sign field, exponent field and mantissa field. The field of sign bit is used to represent the sign of the floating point number. Zero denotes a positive number and one denotes a negative number in this one bit field. The exponent field needs to represent both positive and negative exponents. This field occupies eight bits in the single precision field and eleven bits in the double precision field. The actual exponent is added the value called bias to form the value recorded in this field. The bias value is 127 and 1023 for single precision and double precision respectively. For example, an exponent of zero means that 127 and 1023 is stored in the exponent field for single precision format and double precision format respectively. The mantissa, also called the significant, represents the precision bits of the number. The significand field occupies twenty-three bits and fifty-two bits for the single and double precision format respectively. Whether in single precision or double precision

format, it is composed of an implicit leading bit and fraction bits. In order to maximize the quantity to represent numbers, floating point numbers are stored in normalized form. So the leading digit is assumed to 1 and needs not to represent it explicitly. In other words, the mantissa has effectively twenty-four bits of resolution with twenty-three fraction bits in single precision format. It is similar to the double precision format.

Table 3.5 Format of single and double precision IEEE 754 floating point number

	Sign Field	Exponent Field	Significand Field
Single Precision	1 bit [31]	8 bits [30:23]	23 bits [22:0]
Double Precision	1 bits [63]	11 bits [62:52]	52 bits [51:0]

The summary is described below. First, the sign bit is zero for positive and one for negative number. Second, the exponent field contains 127 added to the true exponent field for single precision format and 1023 added to the exponent field for double precision format. Third, the first bit of the significand is typically assumed to be 1.f, where f is the fraction stored in this field.

The effective range of representing the IEEE 754 floating point number is listed in Table 3.6. In this table the single and double precision format are listed. There are five distinct numerical ranges are not representable in this format. Taking single precision format as an example, the numbers which are not able to present is listed below. The positive number less than  $2^{-149}$ , positive number greater than  $(2-2^{-23}) * 2^{127}$ , zero, negative number less than  $-(2-2^{-23}) * 2^{127}$  and negative numbers greater than  $-2^{-149}$ . They are so called positive underflow, positive overflow, zero, negative underflow and negative overflow respectively. The overflow of the value means it is too large to represent. Underflow is the problem of loss of precision.

Table 3.6 Effective Range of the IEEE 754 floating point number

	Binary Value	Decimal Value
Single Precision	$-(2-2^{-23}) * 2^{127} \sim +(2-2^{-23}) * 2^{127}$	$-10^{38.53} \sim +10^{38.53}$
Double Precision	$-(2-2^{-52}) * 2^{1023} \sim +(2-2^{-52}) * 2^{1023}$	$-10^{308.25} \sim +10^{308.25}$

Finally, special values defined by IEEE 754 standard are introduced. It reserves exponent field values of all zero and all one to denote special values in the floating point values. First, zero will be discussed. Zero is not representable in general format result from the leading one assumption in mantissa field. Zero is defined to be denoted with an exponent field of zero and a fraction field of zero. The positive zero and

negative zero are distinct although they are compared as equal. Then the denormalized number is described. The exponent field is set to all zero and the fraction is non-zero represent a denormalized number. It is not assumed the leading one before the binary point. The representation of value will become  $(-1)^S * 0.f * 2^{-126}$  and  $(-1)^S * 0.f * 2^{-1022}$  for the single precision and double precision format respectively. Third, the value of infinity is denoted with an exponent of all one and a fraction of all zero. The sign bit decides it is positive infinity or negative infinity. Denoting the value of infinity as a specific value is useful because of allowing operations to continue past overflow situations. Operations with infinite values are well defined in IEEE standard. The Last special number is Not A Number (NaN). It is used to represent the value which is not able to represent as a real number. There are two types of NaN such as Quiet NaN (QNaN) and Signaling NaN (SNaN). The most significant bit of fraction field is set for QNaN. The value pops out of an operation when the result is not mathematically defined. The most significant fraction bit is not set for SNaN. It is used to signal an exception.

These features and formats mentioned above is the brief review of the IEEE 754 floating point format standard. The details information of this standard will be listed in the reference list.

Three different types of FPUs are designed and implemented in this thesis. As shown in Fig 3.14 below, the FPU will be integrated with the ALU cluster IP. The data format of the ALU cluster IP is 32 bits so that the IEEE 754 single precision floating point number format is adopted for the design of the floating point operation unit. Three different types of FPUs are described in the following. Considering with the features of the benchmarks and applications, some arithmetic operations of floating point numbers are critical and some are not. In other words, not all of these functional units in the FPUs are operated in the same frequency. Some critical operations need faster clock rate and some need not. The FPU of type 1 includes addition, subtraction and multiplication operations. The FPU of type 2 includes addition, subtraction, multiplication and division operations. The FPU of type3 include division operation only. In most of the media processing applications the division operation is not as common as the addition, subtraction and multiplication operations. So the type1 and type 3 are designed in order to make the critical operations faster and shrink the logic resource needed for the non-critical operations by means of increasing the latencies. The type 2 FPU is also designed for general benchmarks and applications which operations are equally distributed. These FPUs have two input operands and one output results, both of them with 32 bits data width. The operations are decided by the control signal. The details of implementation are introduced in the following chapter.



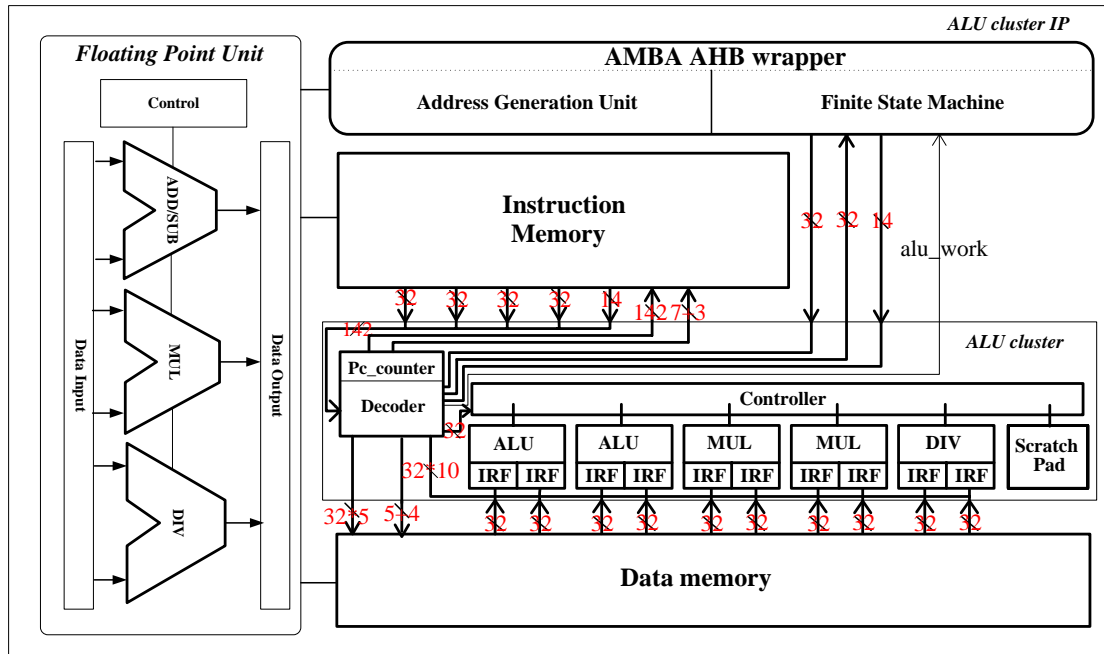


Fig 3.14 An ALU cluster IP with Floating Point Unit Supported Architecture



# CHAPTER 4

## Implementation Results and Performance Evaluation

After the three chapters presented previously, the background, challenges, developmental roadmap and micro-architectures of proposed designs in this thesis are addressed. The circuit implementation results of the proposed designs in this thesis will be discussed in this chapter.

The first section is the demonstration of previous implementation and chip testing results of ALU cluster. Review the implementation results of the ALU cluster, the processing element of ALU cluster intellectual property implemented later. Through testing the silicon-baked chip, the results confirm the correctness of functionality and the architecture is not only feasible but also efficient for media applications.

The second part of this chapter is presented for the implementation and verification results of the designed ALU cluster intellectual property. The features overview of Magnetic RAM and the details of modified architecture included the Magnetic RAM are introduced in this section. Then this section is also described the detail results of taping chip out, circuit verification and chip testing.

The third section of this chapter introduces the circuit implementation and results of floating point operation units for ALU cluster IP. It will be integrated with the ALU cluster IP. The implementation and verification results are summarized in this section. Then the performance evaluation of selected benchmark is estimated to confirm the integration of the floating point unit is efficient and it is the fourth part of the whole chapter.

## 4.1 Implementation and Testing Results of An ALU cluster

The briefly description of the previous design, an ALU cluster, includes the implementation, verification and testing results and related photos. The ALU cluster, the prototype 1 called in the above-presented, is convinced that it can handle media applications expectedly through silicon measurements.

The summary of the manufactured ALU cluster is listed in Table 4.1. UMC 0.18um CMOS technology and cell-based design kit of Artisan are utilized to tape the chip out. The operation frequency of post-layout simulation is 100MHz. The chip size and core size are about  $3 \times 3 \text{ mm}^2$  and  $2.2 \times 2.2 \text{ mm}^2$  respectively. The gate count of this work is 411491. Power consumption of this work is 968.35mW. Besides the logic resources of arithmetic units and control logic, there are total fifteen memory banks used for data and instruction. The instruction memory includes four 32 x 128 single port static RAM (SRAM) and one 14 x 128 single port SRAM. The instruction memory of 128 entries can support output bandwidth of 142 bits per cycle to VLIW instructions. The data memory includes ten 32 x 32 single port SRAM. The data memory of 32 entries can provided the data bandwidth of 320 bits per cycle.

Table 4.1 Implementation Results Summary of ALU cluster

Process	UMC 0.18um CMOS Technology
Library	Artisan SAGE-X Standard Cell Library
Post Layout Clock Rate	100MHz
Chip Size	$2.98 \times 2.98 \text{ mm}^2$
Core Size (without memory)	$2.2 \times 2.2 \text{ mm}^2$ $(1.8 \times 1.2 \text{ mm}^2)$
Gate Count (without memory)	411491 (255669)
Power Consumption (without memory)	968.35mW (312.38mW)
On-Chip Memory	10 block 32 x 32 single port SRAM 4 blocks 32 x 128 single port SRAM 1 block 14 x 128 single port SRAM
Package	CQFP 128
Pad	Input : 47 pins Output : 32 pins Power : 48 pins

These banks of memory are generated by memory compiler with Artisan library. The gate count of this work excluding these memories is 255669 and the power dissipation without these memories is down to 312.38mW. The physical layout of the ALU cluster is shown in Fig 4.1 below. The floorplan and pad assignment are also shown in Fig 4.2.

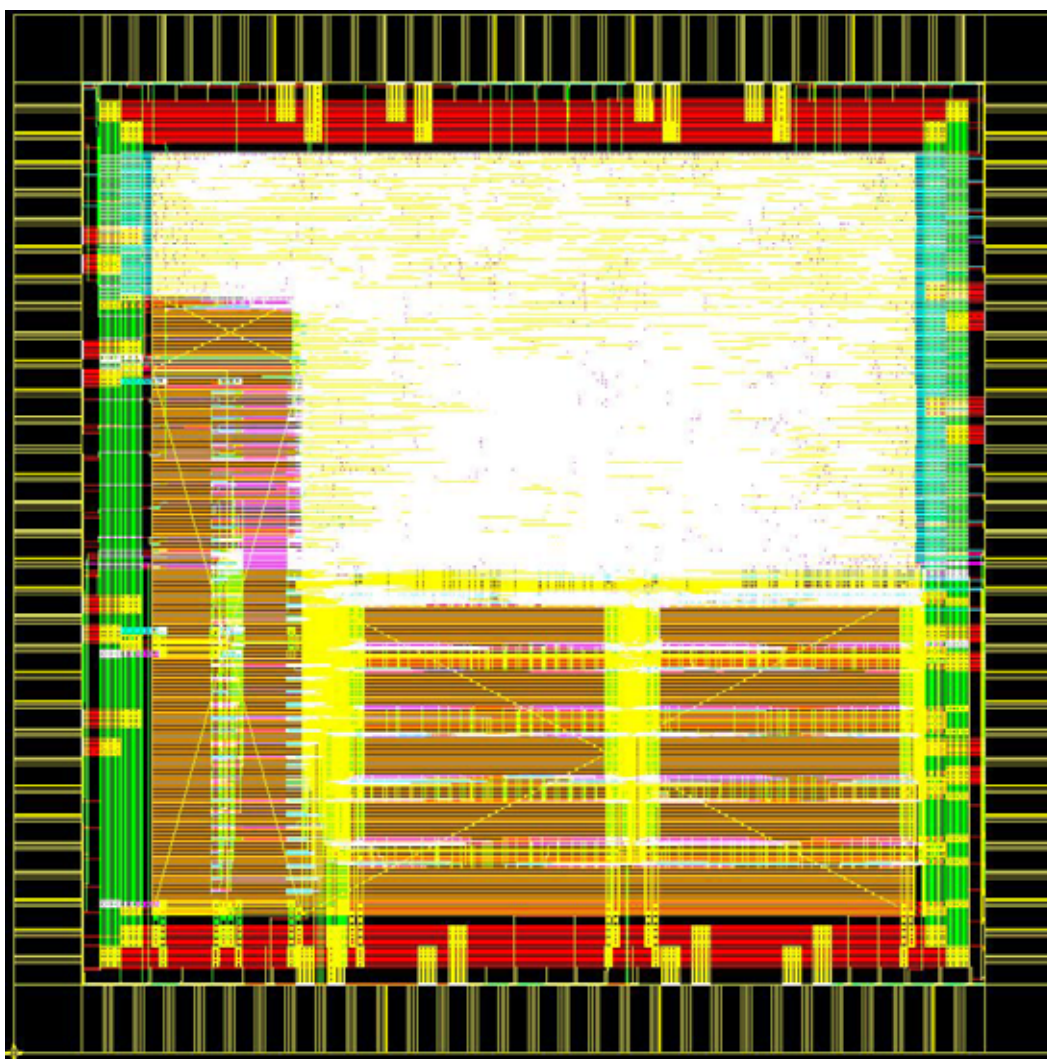


Fig 4.1 Physical Layout of an ALU cluster

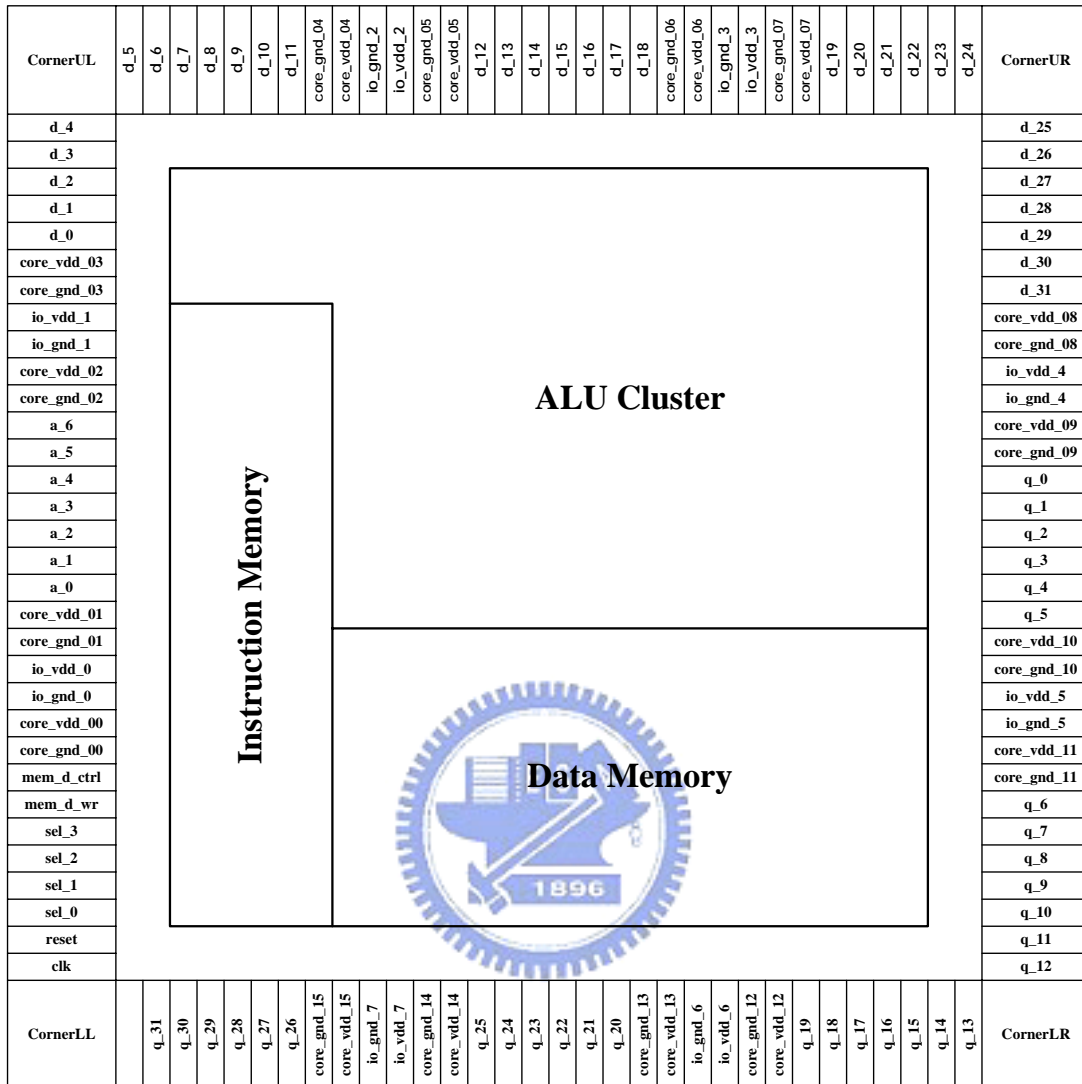


Fig 4.2 Floorplan and Pad Assignment of an ALU cluster

There are total 127 I/O pads, including 47 input pads, 32 output pads and 48 power pads. In addition to the information, the die microphotograph of taped out chip is shown in Fig 4.3. The package used for the manufactured chip is CQFP128 and the photograph of the prototype 1 with package is shown in Fig 4.4.

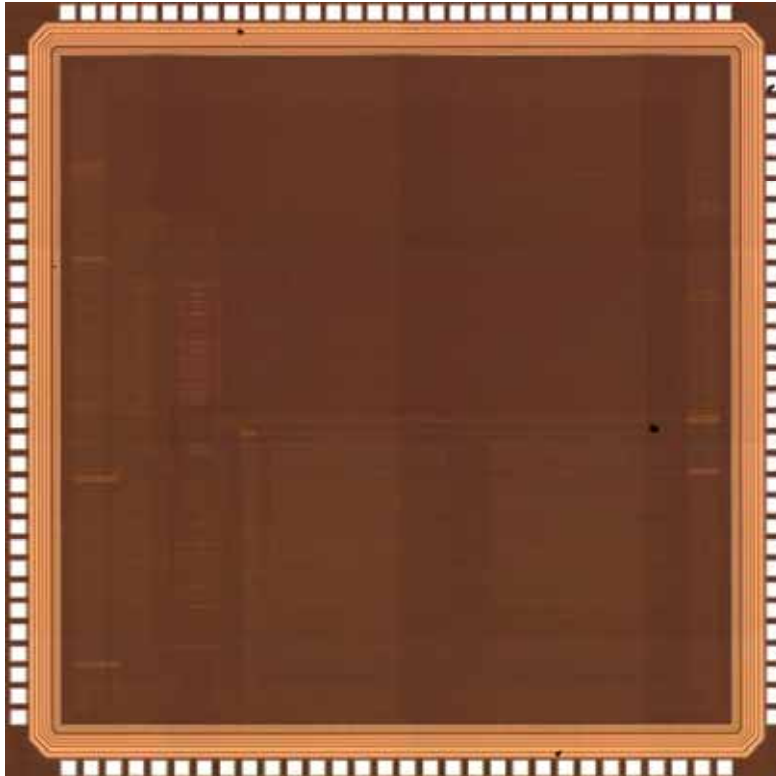


Fig 4.3 Microphotograph of taped out ALU cluster

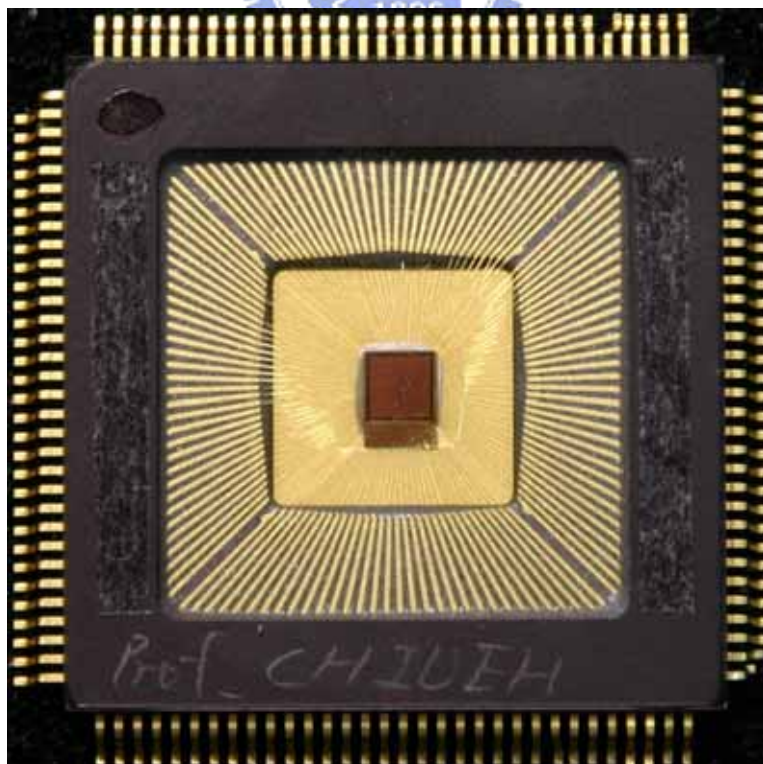


Fig 4.4 An ALU cluster with CQFP128 package

As illustrated in Fig 4.5, the manufactured chip is placed on the PCB board in order to testing. The measurement equipment adopted is Agilent 16902A Logic Analyzer System [27] with Agilent 16720A pattern generator and Agilent 16910A logic analyzer modules. The maximum frequency of signal from the pattern generator is 300MHz and 180MHz when the pattern generator is operated in half channel mode and full channel mode respectively. It is suitable for us to measure the chip.

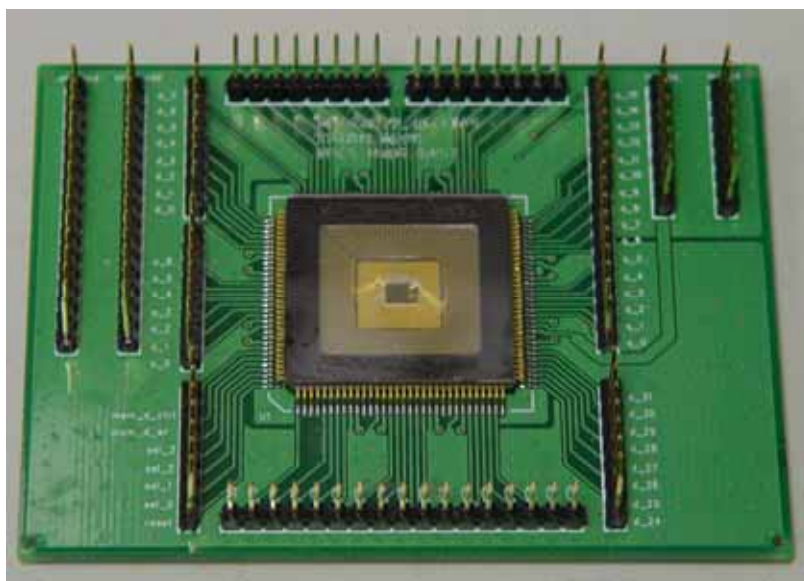


Fig 4.5 An ALU cluster with PCB board

We measure and verify the chip of the ALU cluster mainly in functionality and performance. In the functional testing, the memory testing, the instruction testing and the real program testing are executed. The 16-tap FIR filter is selected as a benchmark for the real program testing and the performance testing. In the memory testing, the benchmarks written to and read from data memory and instruction memory are all one, all zero and mixed interleaving one and zero. These signals are  $32'hFFFFFFF$ ,  $32'h00000000$ ,  $32'hAAAAAAAA$ ,  $32'h55555555$  and mixed the  $32'hAAAAAAAA$  and  $32'h55555555$ . The usage of this kind of benchmark helps us to find out the stuck at zero and stuck at one error. The functionality of instructions is tested by feeding random sequence into all operation units. All instructions of different operation units are gone through at least once to ensure the correctness. These test patterns needed in chip testing necessarily obey the format of the pattern generator.

The functional testing mentioned above including the memory testing, the instruction testing and the real program testing are correct. Thus the performance

testing of the FIT filter system is down to about sixteen to eighteen Mega Hertz. The summaries about testing results mentioned above are listed in Table 4.2. Because of the huge loading of the probe lead set, it is more than five times slower than the post layout clock rate. It is hard to measure the true performance of the chip unless connecting the pod of pattern generator to the PCB directly.

Table 4.2 Testing results summaries of ALU cluster

Testing Items		Results
Read/Write Memory	Data Memory	Correct
	Instruction Memory	Correct
Instruction Functionality	ALU0	Correct
	ALU1	Correct
	MUL0	Correct
	MUL1	Correct
	DIV	Correct
FIR filter system	Correct with operating in 16~18 MHz	

## 4.2 Verification and Implementation Results of An ALU cluster Intellectual Property

In this section, an ALU cluster Intellectual Property (IP) is designed and implemented. As mentioned in previous chapter, there is an AMBA AHB wrapper in the ALU cluster IP different from the ALU cluster. First, the modified ALU cluster IP architecture will be presented and the introduction of Magnetic RAM is described. The taped out chip is concert with the Magnetic RAM (MRAM) developed by Industrial Technology Research Institute (ITRI). Then we will introduce the implementation of the ALU cluster Intellectual Property. Finally the circuit verification and chip testing are introduced in the last two sub-sections in the paragraph. The details are described in the following sub-sections.

### 4.2.1 An ALU cluster IP with Magnetic RAM

An ALU cluster IP with Magnetic RAM is the extended version of the ALU cluster IP. First we will introduce briefly the characteristics of MRAM. Then the architecture of the ALU cluster is slightly modified to adapt the features of using MRAM as the data memory of the IP. Finally the implementation and verification results of the manufactured chip are discussed.



### 4.2.1.1 Introduction of Magnetic RAM

Three types of memory such as static RAM (SRAM), dynamic RAM (DRAM) and Flash occupy the most proportion of current market. However, they have their own drawback respectively. SRAM and DRAM have high data accessing speed, but they are volatile when power is turned off and the power consumption of these two types of memory is high. These drawbacks can be solved when using the Flash. It is non-volatile and the power dissipation is low. But the accessing speed of Flash is low and the lifetime of reading and writing is limited.

Magnetic RAM (MRAM) is the innovation of these types of memory [28]. It adopts the magnetic tunnel junction (MTJ) and MOSFET as the mechanism to form the memory cell [29]. It collects the pros, such as non-volatile, low power, high accessing speed, high cell density and strong radiation resistance, of current existent memories. In addition, the MRAM has the advantages such as process compatible and long lifetime of writing and reading. Because the MRAM is manufactured by the metal layer so it is compatible to CMOS technology and without extra overhead. And its lifetime of data accessing is much higher than the conventional non-volatile memory such as Flash.

MRAM applications currently targets to mobile system, smart card, radiation hardened military applications, database storage, RFID and MRAM element in FPGA. These include both standalone and embedded memory applications. As these above-mentioned pros and wide application in modern life, MRAM will become a chief tendency of new generation memory system.

### 4.2.1.2 Modified ALU cluster IP for Magnetic RAM

The architecture of ALU cluster IP with SRAM as the data memory must be modified to adopt MRAM as the data memory because of the interface of MRAM and SRAM is not the same and the data bandwidth between these two memories is also different. In order to connect MRAM to our IP, an extra load store unit (LSU) must be added to solve the issue presented above. The modified architecture is shown in Fig 4.6. The instruction format is changed slightly from 142 bits to 143 bits. The additional bit is used to control the mode of the ALU cluster IP. If the additional bit is not set, the IP will execute the applications normally. When the additional bit is set, the data in IRF and MRAM could be accessed separately which is based on the executing instruction. The data bandwidth between IRF and MRAM is restricted by

MRAM. The bandwidth support by the IRF is also modified to support byte transfer. This is also suitable for the AHB wrapper since it is designed for byte, half word, and word access in little endian manner the same as non-modified IP.

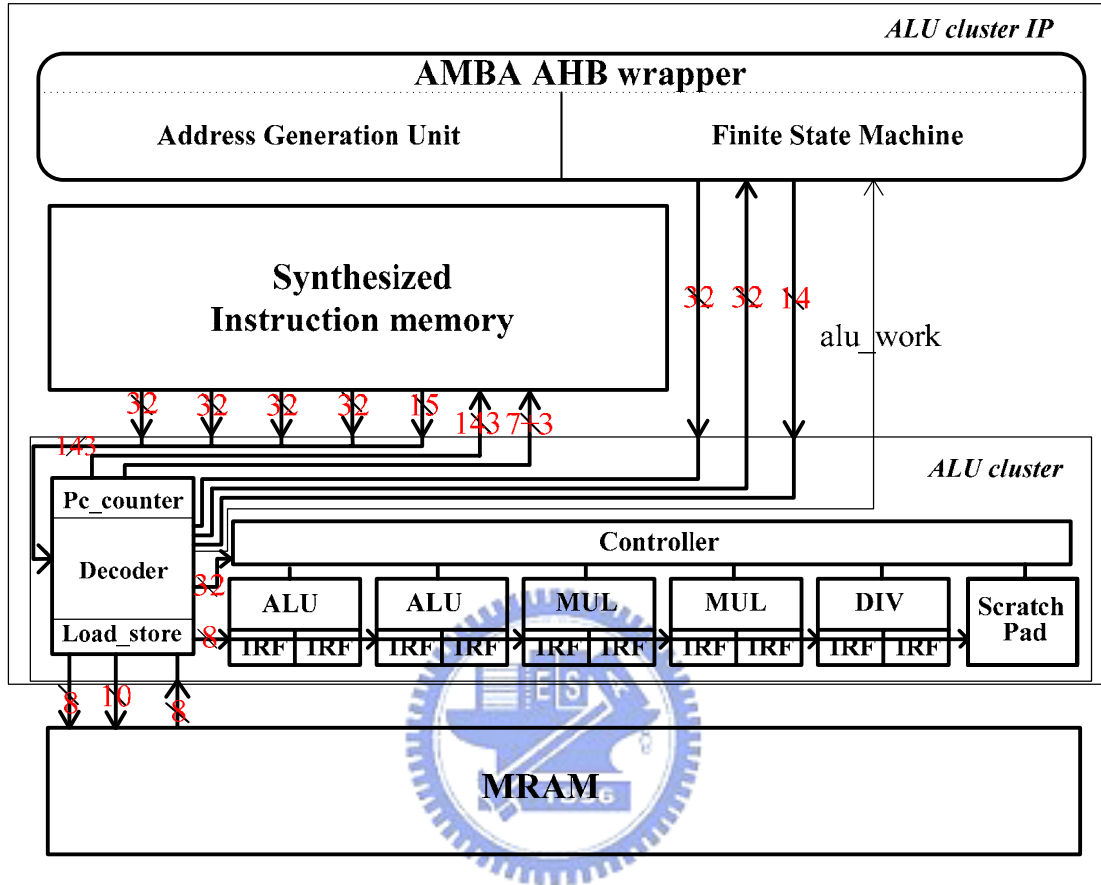


Fig 4.6 Modified ALU cluster IP architecture for MRAM

## 4.2.2 Implementation Results

The summary of implementation characteristics are listed in Table 4.3. The proposed ALU cluster IP is implemented with cell-based design flow and taped out using TSMC 0.15um CMOS technology. Synopsys computer aided design (CAD) flow is adopted to accomplish this chip. The post-layout operation frequency of an ALU cluster IP is 100 MHz. The chip size, core size and gate count are about 3.9x3.9 mm<sup>2</sup>, 3.0x3.0 mm<sup>2</sup> and 0.2 million, respectively. The physical layout and pad assignment are shown in Fig 4.7 and Fig 4.8 respectively.

Table 4.3 Summary of Implementation Characteristics

<b>Process</b>	<b>TSMC 0.15um</b>
<b>Post-layout Clock Rate</b>	<b>100 MHz</b>
<b>Chip Size</b>	<b>3.91 x 3.90 mm<sup>2</sup></b>
<b>Core Size</b>	<b>2.98 x 2.98 mm<sup>2</sup></b>
<b>Gate Count</b>	<b>267,473</b>
<b>On-chip memory</b>	<b>Instruction Memory : synthesized Data Memory : MRAM</b>
<b>Package Type</b>	<b>COB(PGA256)</b>
<b>Pad</b>	<b>Input: 34 pins Inout : 32 pins Output: 24 pins Power: 40 pins</b>

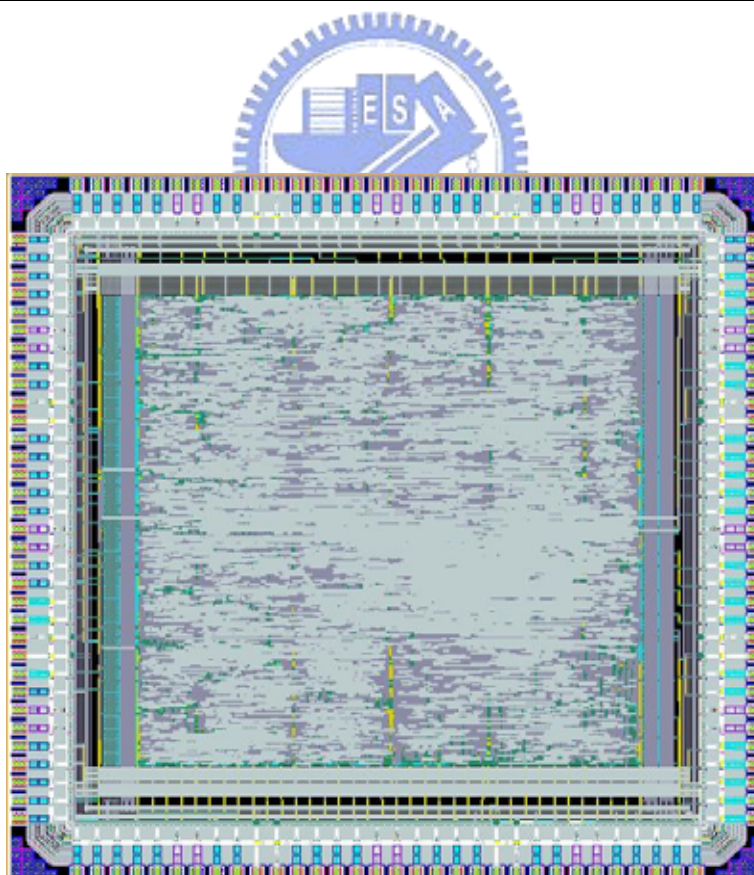


Fig 4.7 Physical Layout of an ALU Cluster IP

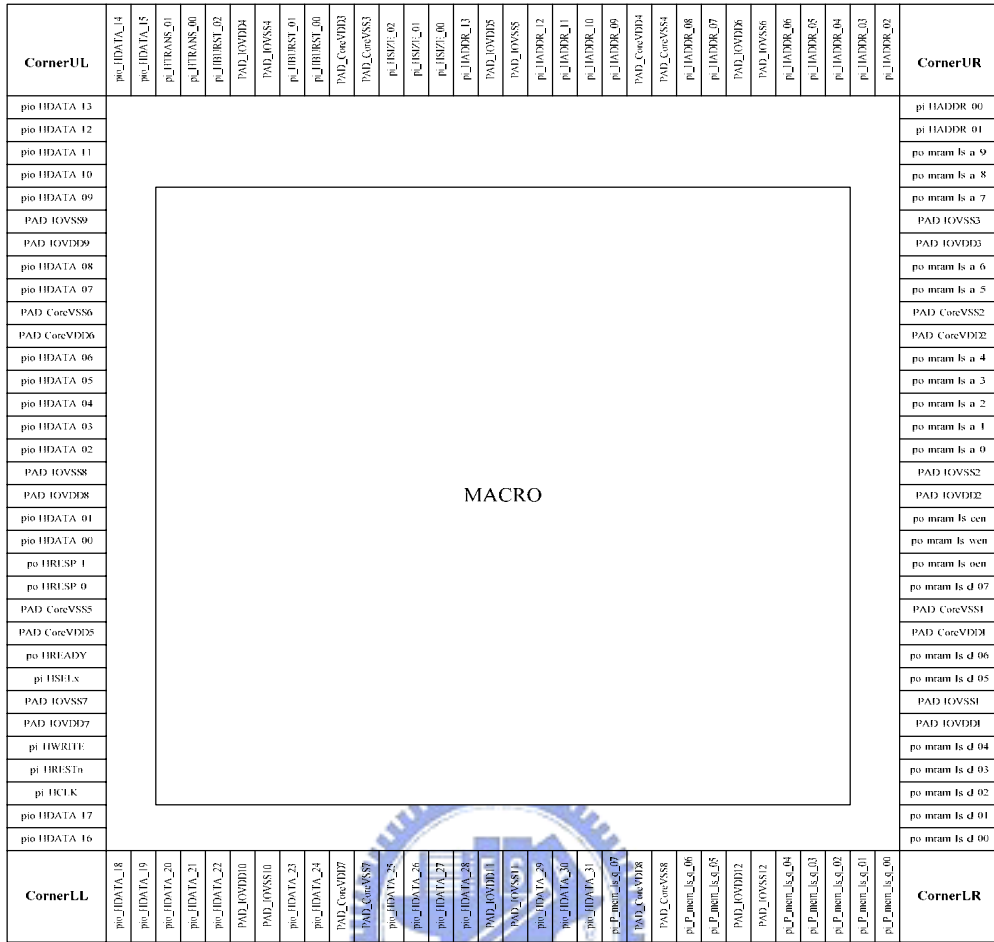


Fig 4.8 Pads Assignment of an ALU Cluster IP

There are total 130 pads, where 34 input pads, 24 output pads, 32 inout pads and 40 power pads in this design. In addition, the die microphotograph of taped out chip is shown in Fig 4.9. The selected package for the manufactured die is PGA256. The prototype with package is shown in Fig 4.10. The definitions of I/O ports are listed in Table 4.4.



Fig 4.9 Die Microphotograph of Taped Out Chip

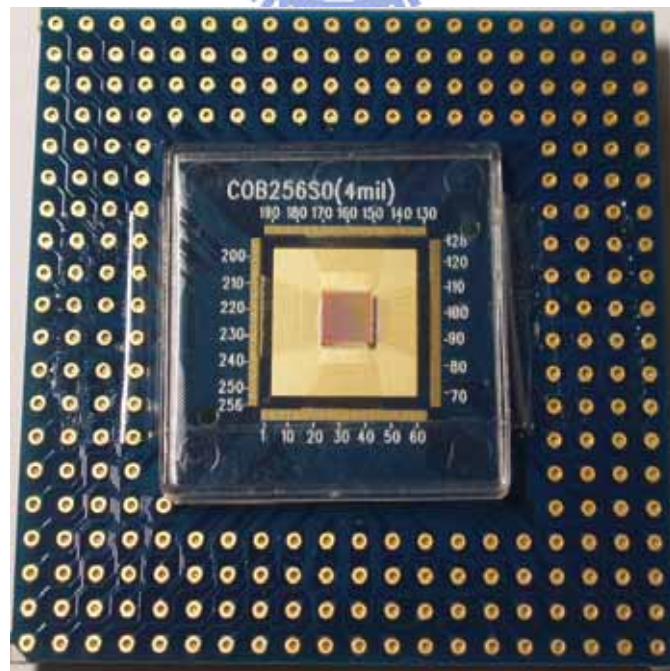


Fig 4.10 Photograph of Prototype with Package

Table 4.4 The Definitions of I/O the ports

I/O Port Name	Input/Output/In out	Signal Description
HCLK	Input	The clock signal provides for designed chip
HADDR	Input	This is a 14-bits input used to specify the address of instruction memory and data memory
HSEL <sub>x</sub>	Input	The select signal from the arbiter of AHB bus to enable bus slave to work. It will be logical low at all execution stage.
HWRITE	Input	The signal indicates a write transfer at logical high and a read transfer at logical low.
HTRANS	Input	The 2-bits signal determines the transfer type of AHB protocol including IDLE, BUSY, NONSEQ and SEQ.
HSIZE	Input	The 3-bits signal used to determine the size of transfer.
HBURST	Input	This 3-bits signal indicates which type of burst mode is used. The burst may be either incrementing or wrapping and four, eight and sixteen beat bursts are supported.
HREST <sub>n</sub>	Input	The reset signal provides for this chip.
mem_ls_q	Input	The 8-bits signal used to receive data from data memory, Magnetic RAM, which is not embedded in the chip.
HREADY	Output	The signal uses to indicate whether the transfer has finished on the bus or not. Logic high means it is finished and logic low means that the transfer need to extend
HRESP	Output	2-bits signal response the status of a transfer. OKAY, ERROR, RETRY and SPLIT are provided.

mram_ls_d	Output	The signal is an 8-bits data output to the data memory which is not embedded in this chip.
mram_ls_a	Output	The 10-bits width output. Used to specify the address of the external data memory.
Mram_ls_cen	Output	These three signals are used to control the status of the external data memory, such as disable, read, write and selected disable mode, by different combination of these signals.
Mram_ls_wen		
Mram_ls_oen		
HDATA	Inout	The 32-bits inout signal. These signals receive data from input ports to compute and output the calculated results outside the chip.
IOVDD & IOVDD	Power	The power supply provides for the core of this chip. There are 12 pairs of power supply.
CoreVDD & CoreVSS	Power	The power supply provides for the IO Pads. There are 8 pairs of power supply.

### 4.2.3 Circuit Verification

The popular operation of multimedia processing applications, the finite impulse response (FIR) filter system [30], is chosen as the benchmark of the ALU cluster IP. The benchmark used to simulate and verify the proposed IP is 16-tap FIR filter system. The media applications could be expressed as the stream programming model that would be fit the features of the ALU cluster IP. In modern media and DSP applications, FIR filtering is one of the most popular and widely operation applied, such as matched filtering, pulse shaping and equalization, etc. This selected benchmark is suitable for functional verification of one dimensional architecture needed repeat and high percentage of addition and multiplication. .

A brief review of FIR filter system is introduced below. The equation of input and output relationship of linear time invariant FIR filter can be describe in Equation 4.1.

$$y[n] = \sum_{k=0}^{M-1} b_k * x[n-k] \quad \text{Equation 4.1}$$

As shown in the equation,  $M$  represents the length of the FIR filter,  $b_k$  represents the coefficients and  $x[n-k]$  denotes the data sampled at time instance  $n-k$ . The output  $y[n]$  is the response to the instance time  $n$ . As illustrated in Fig 4.11, the coefficients  $b_k$  of the sixteen-tap Kaiser window FIR bandpass filter and the exponential function with ten sampling points as the input data are figured. The usage of Mathworks Matlab helps us to simulate the correct results in advance. The results are illustrated in Fig 4.12.

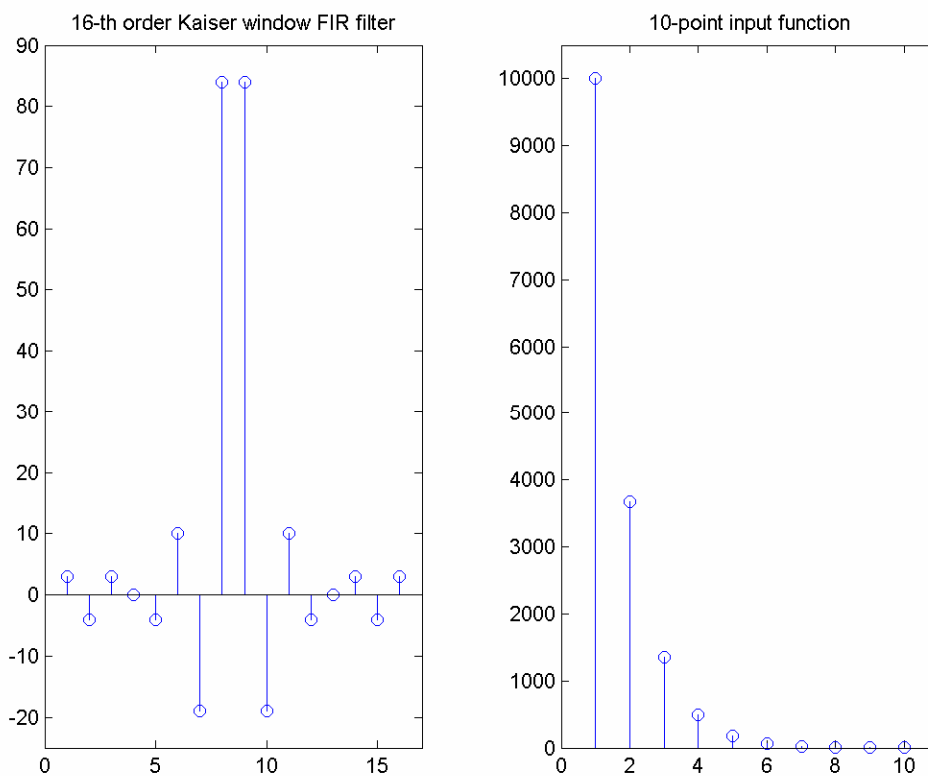


Fig 4.11 The input function and coefficients of the FIR filter system



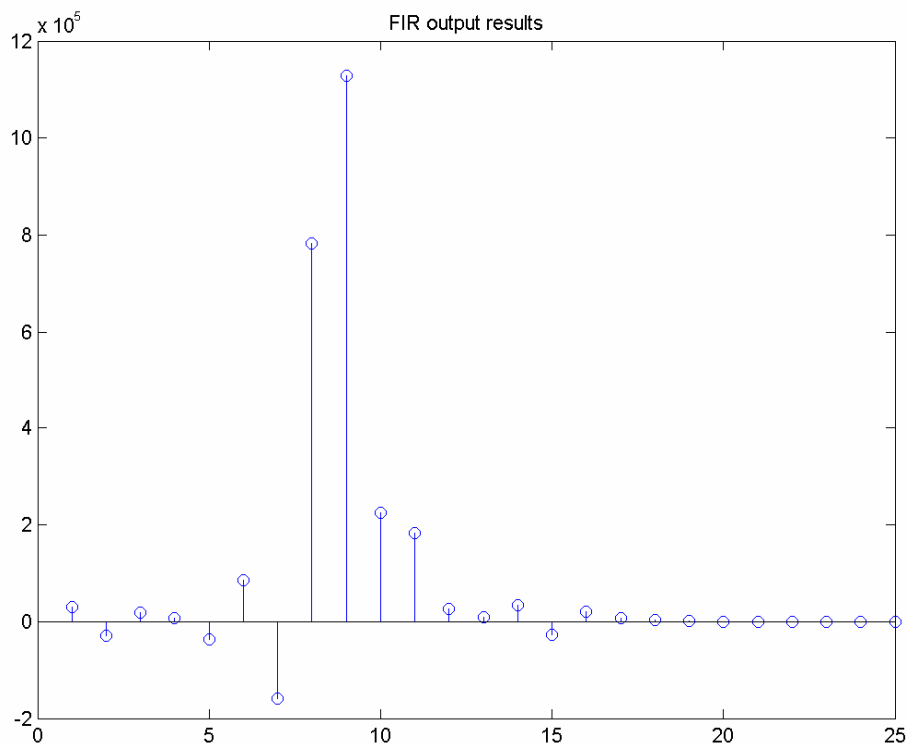




Fig 4.12 Output results of the FIR filter system

After simulating the 16-tap FIR filter system with Mathworks Matlab, the benchmark is ported into the implemented design as circuit verification. The circuit verification is executed by post-layout simulation with the taped-out chip introduced above. Due to the imperfect library of the CMOS technology, the simulation results addressed here are from the ALU cluster IP hard macro. The results of the post-layout simulation are listed from Fig 4.13(a) to Fig 4.13(e) continuously. The post-layout simulation is based on TSMC 0.15um CMOS technology. The circuit of the proposed design shows that it works correctly at clock rate of 100 MHz after comparing between the final result from Matlab and the results of post-layout simulation. It reveals that the functionality is correct exactly and IP works correctly.



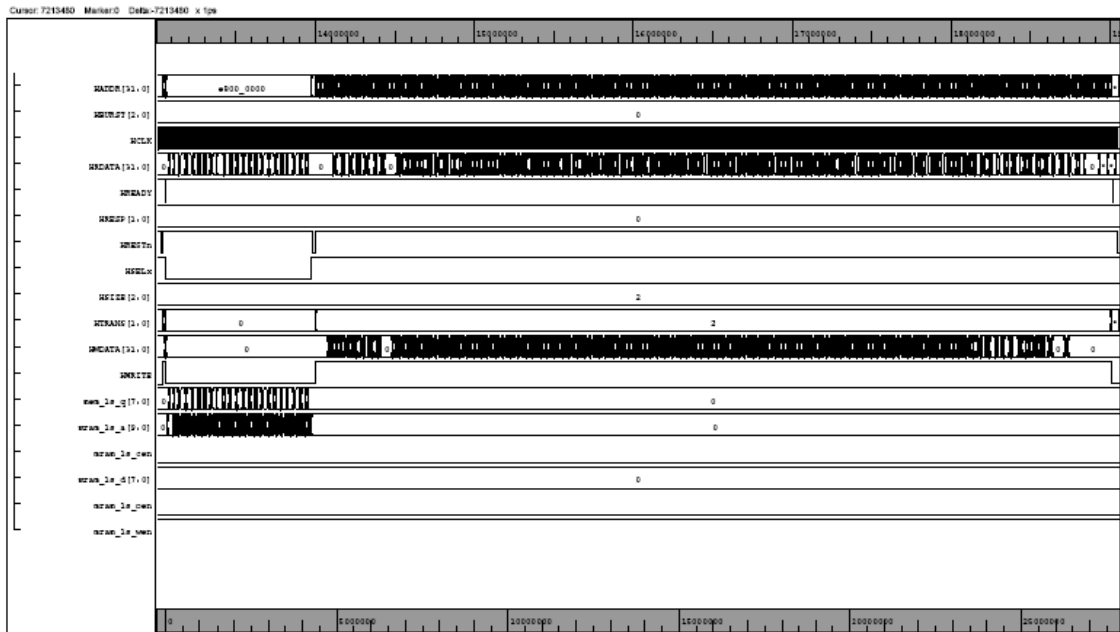


Fig 4.13(c) Post-Layout Simulation Results of an ALU cluster IP ( )

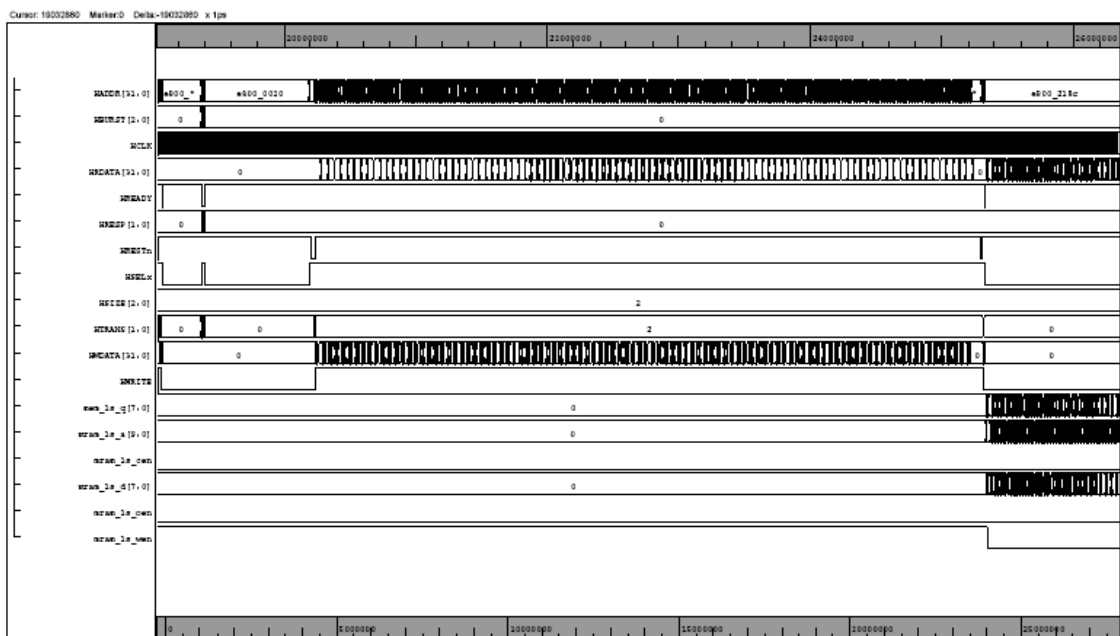


Fig 4.13(d) Post-Layout Simulation Results of an ALU cluster IP ( )



The testing equipments adopted are Agilent 16902A Logic Analyzer System with Agilent 16720A pattern generator and Agilent 16910A logic analyzer modules as shown in Fig 4.15. In order to avoid the performance degradation results from the pods of pattern generator modules, pods of pattern generator are directly connected to the PCB combined the chip. A diagram is illustrated in Fig 4.16.

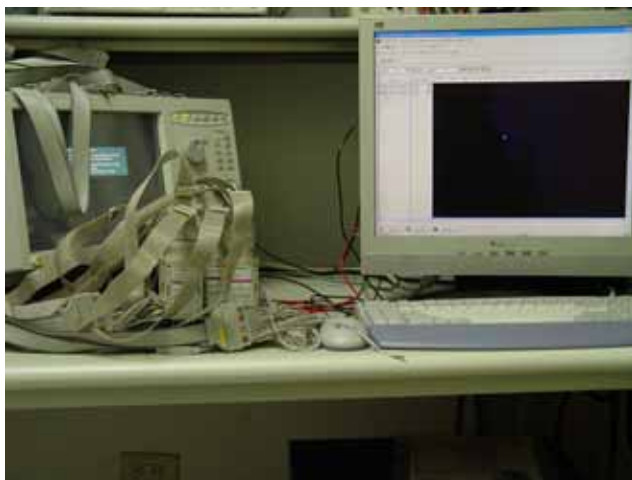


Fig 4.15 Testing Equipments – Logic Analyzer System



Fig 4.16 Connection between the Chip and Testing Equipments

Chip testing is divided into two parts such as functional measurement and performance measurement. The performance measurement adopts the 16-tap FIR filter system presented previously as the benchmark. And the functional measurements include basic functions, writing to IRF and memory and reading from IRF and memory. While verifying the chip, several phenomenons are revealed. First, the input signals from pattern generator modules into the chip can not be sent

correctly. After measuring the signal through an oscilloscope, the phenomenon that the signals have irregular voltage values near the threshold voltage is revealed. Besides, the peripheral signals on the PCB are also measured and the phenomenon is also still existent. Second the control signal for the MRAM such as the signal of chip enable is toggled irregular and it disagrees with the results of post-layout simulation. Not only the chip enable signal but also some other signals also has the same phenomenon. These are two main phenomenon observed from the chip testing and they may cause the errors of the measurement. Chip testing is progressing and the errors will be solved and discussed.

### 4.3 Circuit Implementation and Results of Floating Point Units for the ALU cluster IP

In this section, the implementation results of the FPUs described above will be introduced. These FPUs are implemented as hard macros with cell-based design flow. The macros can be used to integrated with the ALU cluster IP and provide efficient floating points operations ability. The synthesis results and the results of Auto Place and Route (APR) are discussed. The circuit verification results executed through post-layout simulation are also listed in this section too.

The floating point units are synthesized with Synopsys Design Compiler and the physical layouts of these FPU macros are finished by means of the Synopsys Astro. The TSMC 0.18um CMOS technology and Artisan SAGE-x Standard Cell Library are adopted for implementing these FPUs. As mentioned above, three types of FPUs are designed. Type 1 FPU includes the floating point operations for addition, subtraction and multiplication. It operates in 75 MHz of post simulation frequency. The gate count and area are 23,298 and 0.415 mm<sup>2</sup> respectively. The area utilization and power consumption are 0.9 and 10.85 mW respectively. The physical layout of the type 1 macro of the FPU is shown in Fig 4.17. As shown in Fig 4.18, the macro of type 2 FPU is implemented completely. It provides the floating point operations including addition, subtraction, multiplication and division inside. The post simulation clock rate of type 2 FPU is 25 MHz. The gate count and area are 31,331 and 0.529 mm<sup>2</sup> respectively. The area utilization, the same as type 1 FPU, is 0.9. The power dissipation of type 2 FPU is 4.60 mW. Type 3 FPU, including the division floating point operation only, is in order to collocate with type 1 FPU to provide the same operations as type 2 FPU. So the type 3 FPU is implemented with the 25 MHz clock rate of post simulation in spite of it is not the fastest operation frequency it can achieve. The gate count and area are 24,931 and 0.396 mm<sup>2</sup>. The area utilization, the

same as type 1 and type 2 FPU, is 0.9 and the power dissipation is 6.59 mW. The physical layout of type 3 FPU macro is shown in Fig 4.19 below. The summary of these results are listed in Table 4.5

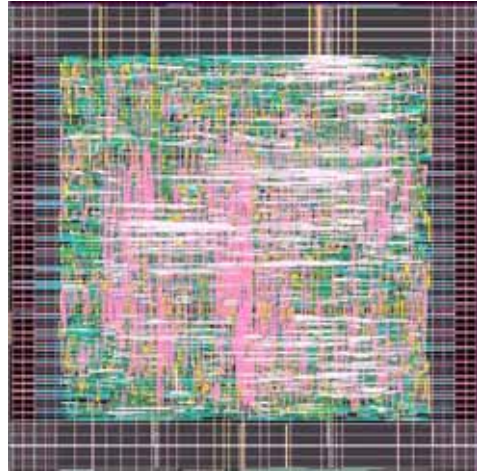


Fig 4.17 Physical Layout of the Type 1 FPU macro

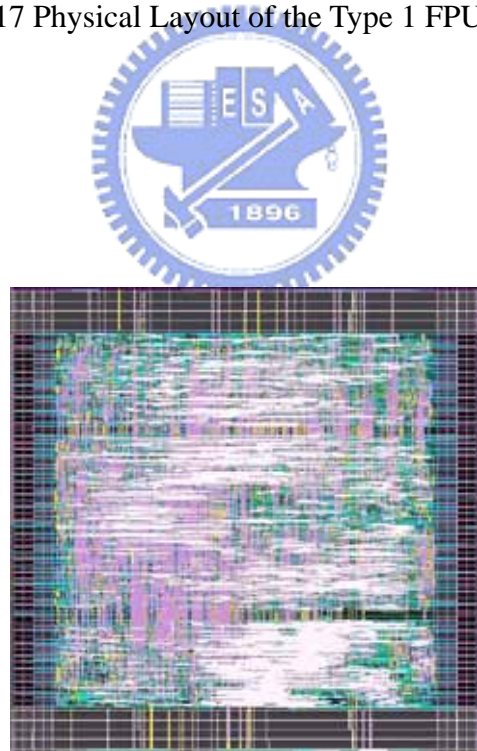


Fig 4.18 Physical Layout of the Type 2 FPU macro

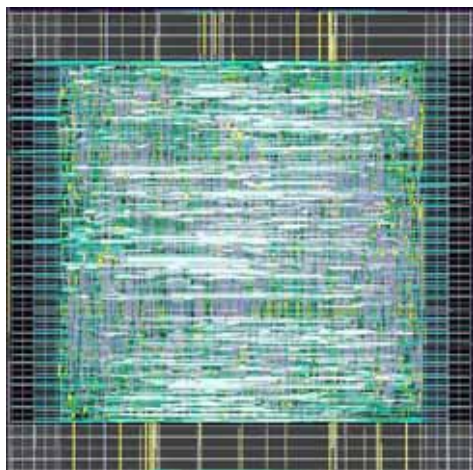


Fig 4.19 Physical Layout of the Type 3 FPU macro

Table 4.5 Summary of the Implementation Results

Floating Point Unit	Type 1 (ADD, SUB, MUL)	Type2 (ADD, SUB, MUL, DIV)	Type 3 (DIV only)
Technology	TSMC 0.18um	TSMC 0.18um	TSMC 0.18um
Cell Library	Artisan SAGE-X™	Artisan SAGE-X™	Artisan SAGE-X™
Post-layout Simulation Clock Rate	75 MHz	25 MHz	25 MHz
Area Utilization	0.9	0.9	0.9
Power Consumption (mW)	10.85	4.60	6.59
Gate Count	23,298	31,331	24,931
Area (mm <sup>2</sup> )	0.415 mm <sup>2</sup>	0.529 mm <sup>2</sup>	0.396 mm <sup>2</sup>

The circuit verification results are listed below. Post-layout simulation are performed with the TSMC 0.18um CMOS technology and library environment. The post-layout simulation results for type 1 FPU are shown in Fig 4.20 and Fig 4.21. They are the full view and the interception of whole simulation periods respectively. The same as type 1, the verification results of type 2 FPU are shown in Fig 4.22 and Fig 4.23 and they are full view and a portion of all periods respectively. Eventually the post-layout simulation results of type 3 FPU are illustrated in Fig 4.24 and Fig 4.25. As described previously, they are also full view and interception of whole simulation periods respectively. These outcomes promise that these hard macros work correctly corresponding to their own clock rate of post-layout simulation.



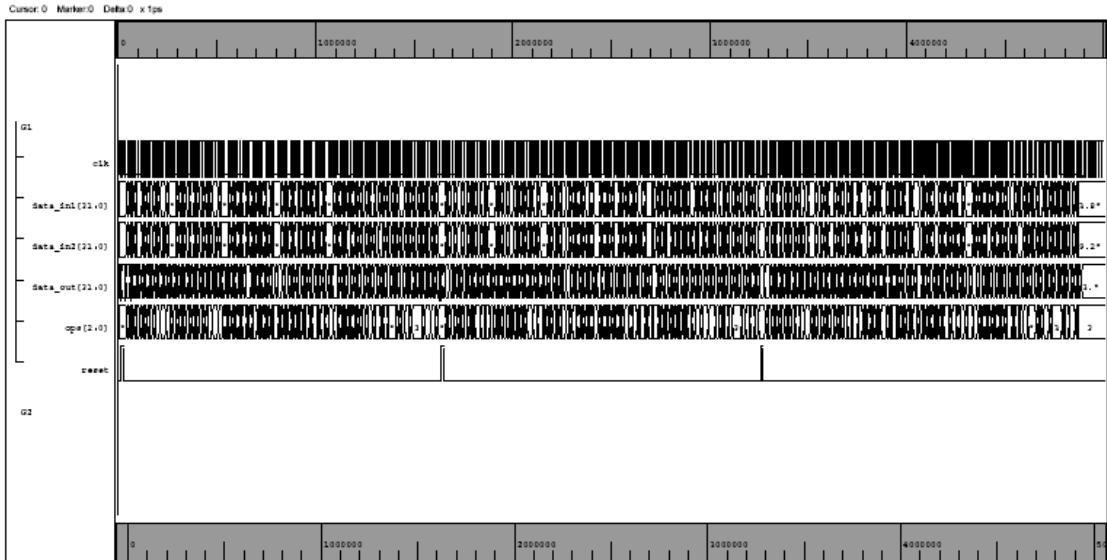


Fig 4.20 Full View of Post-Layout Simulation Results for Type 1 FPU

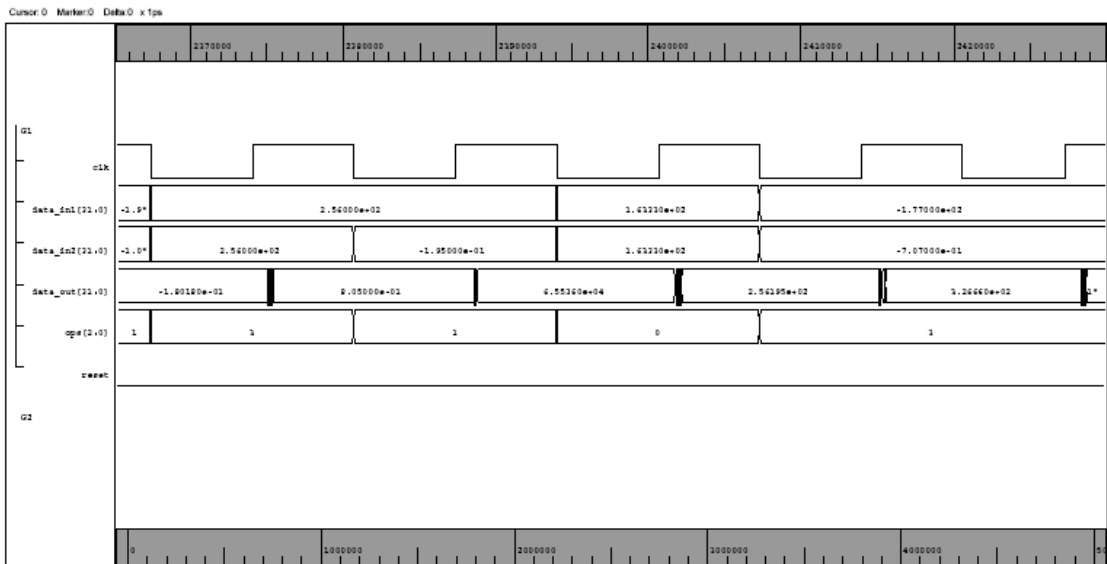


Fig 4.21 Interception of Post-Layout Simulation Results for Type 1 FPU

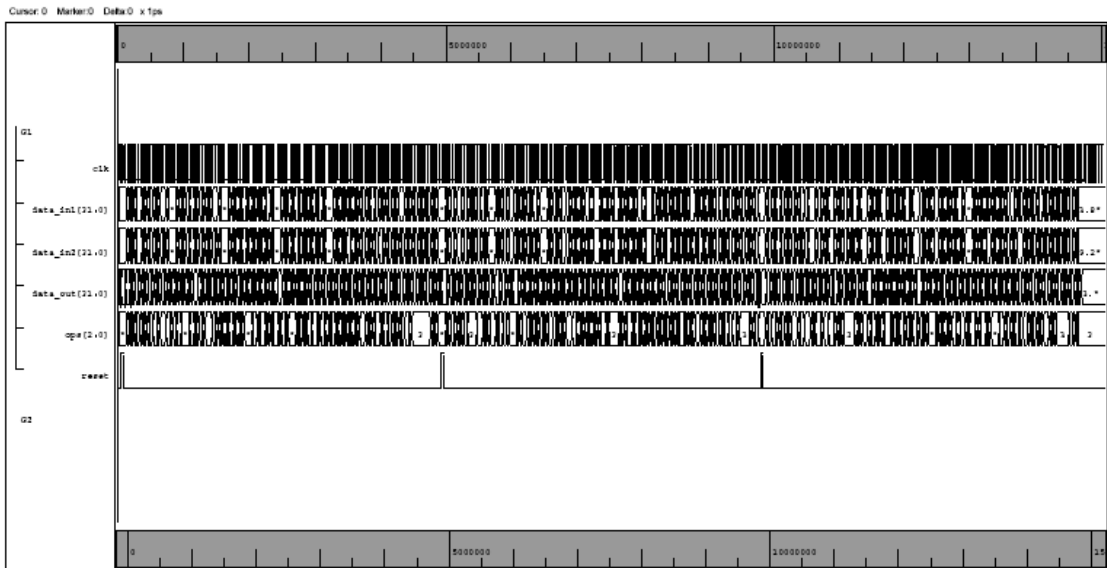


Fig 4.22 Full View of Post-Layout Simulation Results for Type 2 FPU

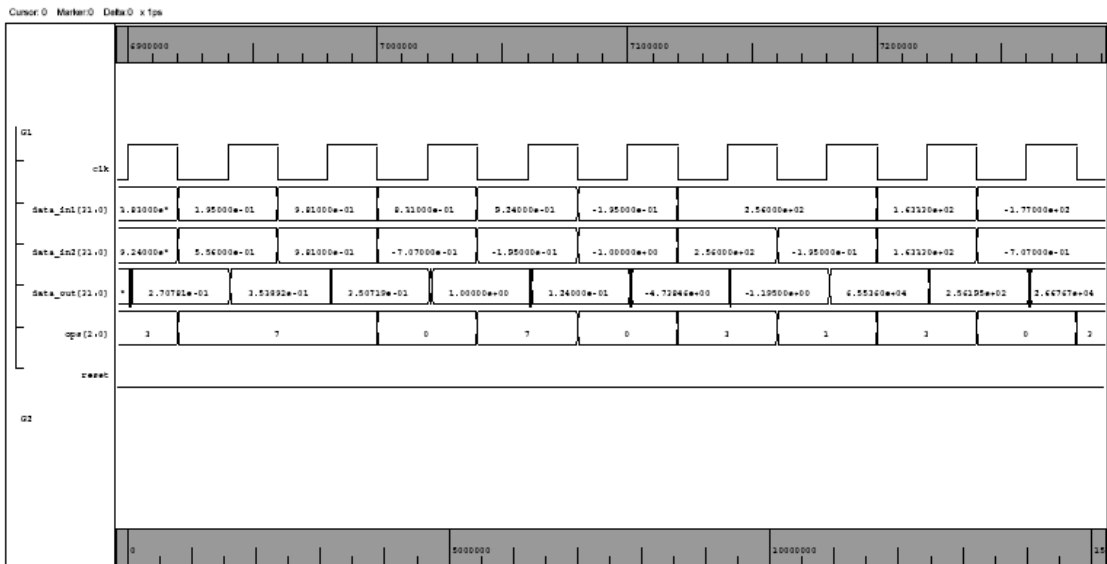


Fig 4.23 Interception of Post-Layout Simulation Results for Type 2 FPU

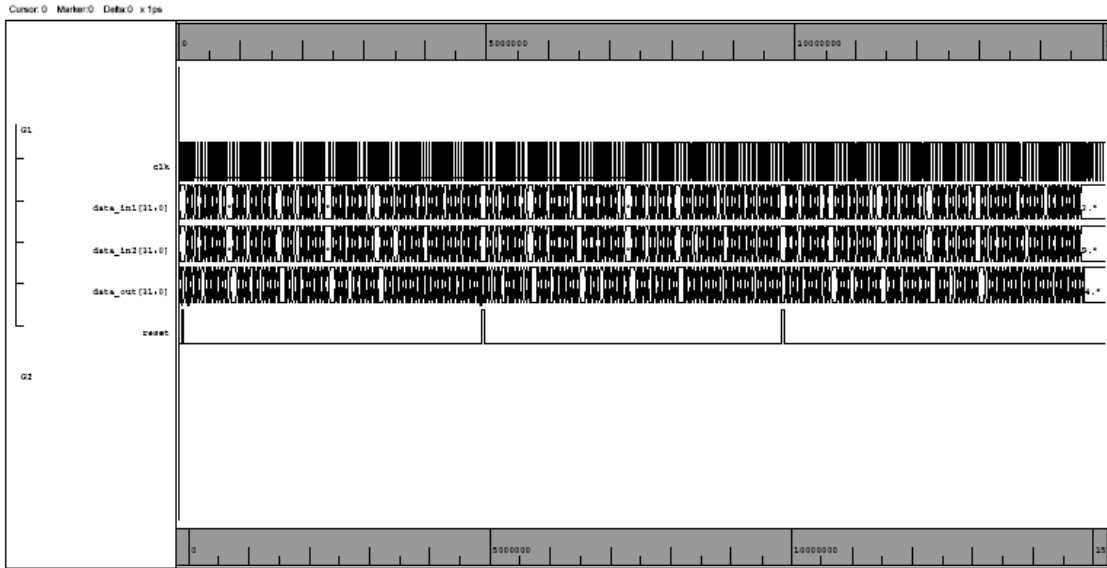


Fig 4.24 Full View of Post-Layout Simulation Results for Type 3 FPU

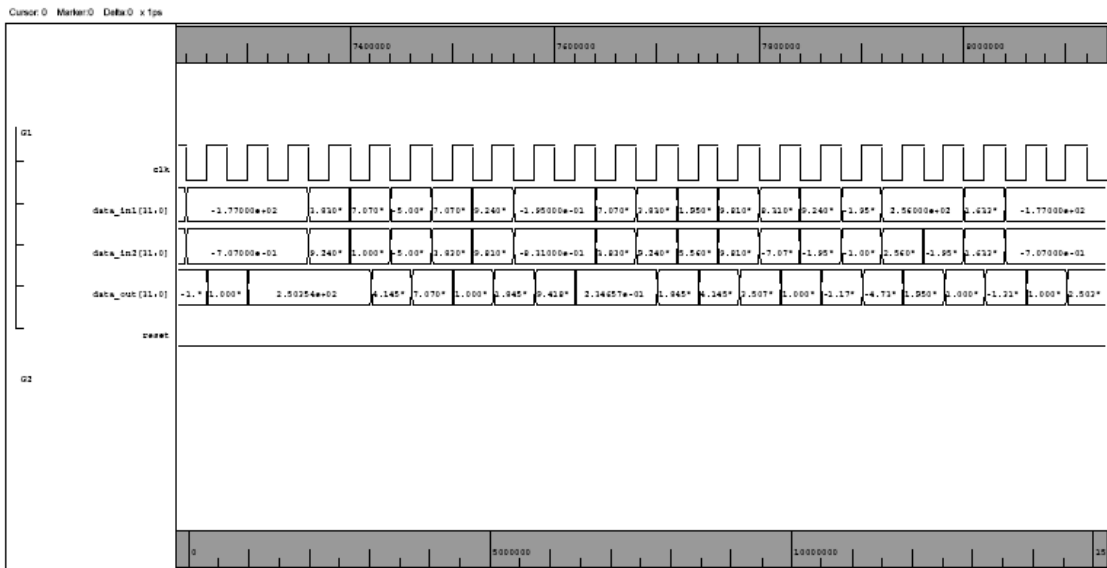


Fig 4.25 Interception of Post-Layout Simulation Results for Type 3 FPU

#### 4.4 Performance Evaluation and Comparison

In this section, the performance of floating point operations will be evaluation and comparison. The benchmark used to evaluation is the Fast Fourier Transform (FFT) commonly used in media processing applications. There are two target architectures used to evaluate the performance of floating point operations and compare the performance each other. These two parts mentioned above are discussed in the following.

### 4.4.1 Selected Benchmark

In this thesis, 32-points Fast Fourier Transform is selected as the benchmark. The FFT is an efficient algorithm for computation of the Fourier Transform. There are three reasons that the FFT is selected. First, the FFT is the most often used operations in the multimedia applications or signal processing applications. The second reason is taken as an example that executing on the streaming programming model in paper [18]. Third, the applications involved the FFT usually need to handle the floating point numbers operations. It is well-match for the performance evaluations presented in this thesis.

The Split-Radix FFT (SRFFT) algorithm is adopted to form the benchmark [31] [32]. An inspection of the decimation in frequency flowchart of FFT shows that the even terms and the odd terms of the Discrete Fourier Transform (DFT) can be computed independently. It is quite clear that the radix-2 algorithm is better for the even terms and the radix-4 algorithm is better for the odd terms of the DFT. So the split-Radix FFT (SRFFT) algorithm which reduces the number of computations exploits the idea of using radix-2 and radix-4 algorithms mixed into the same FFT algorithm.

As mentioned above, the FFT algorithm is decomposed into even terms and odd terms to compute independently. The radix-2 decimation in frequency FFT algorithm used for the even numbered samples of the N-points DFT are given in below.

$$X(2k) = \sum_{n=0}^{N/2-1} \left[ x(n) + x\left(n + \frac{N}{2}\right) \right] W_{N/2}^n, \text{ where } k = 0, 1, \dots, \frac{N}{2} - 1, \quad W_N = e^{-j2\pi/N}$$

The odd-numbered samples  $[X(2k+1)]$  of the DFT need to pre-multiplication of the input sequence with  $W_N^n$ . The radix-4 decimation in frequency FFT algorithm used for the odd-numbered samples of the N-points DFT are given in the two equations below.

$$X(4k+1) = \sum_{n=0}^{N/4-1} \left\{ \left[ x(n) - x\left(n + \frac{N}{2}\right) \right] - j \left[ x\left(n + \frac{N}{4}\right) - x\left(n + \frac{3N}{4}\right) \right] \right\} W_N^n W_{N/4}^{kn}$$

$$X(4k+3) = \sum_{n=0}^{N/4-1} \left\{ \left[ x(n) - x\left(n + \frac{N}{2}\right) \right] + j \left[ x\left(n + \frac{N}{4}\right) - x\left(n + \frac{3N}{4}\right) \right] \right\} W_N^{3n} W_{N/4}^{kn}$$

The length-N DFT will be obtained by using the N Split-Radix FFT algorithm. The benchmark adopted is the length 32 Split-Radix FFT and its flowchart is shown in Fig 4.26.

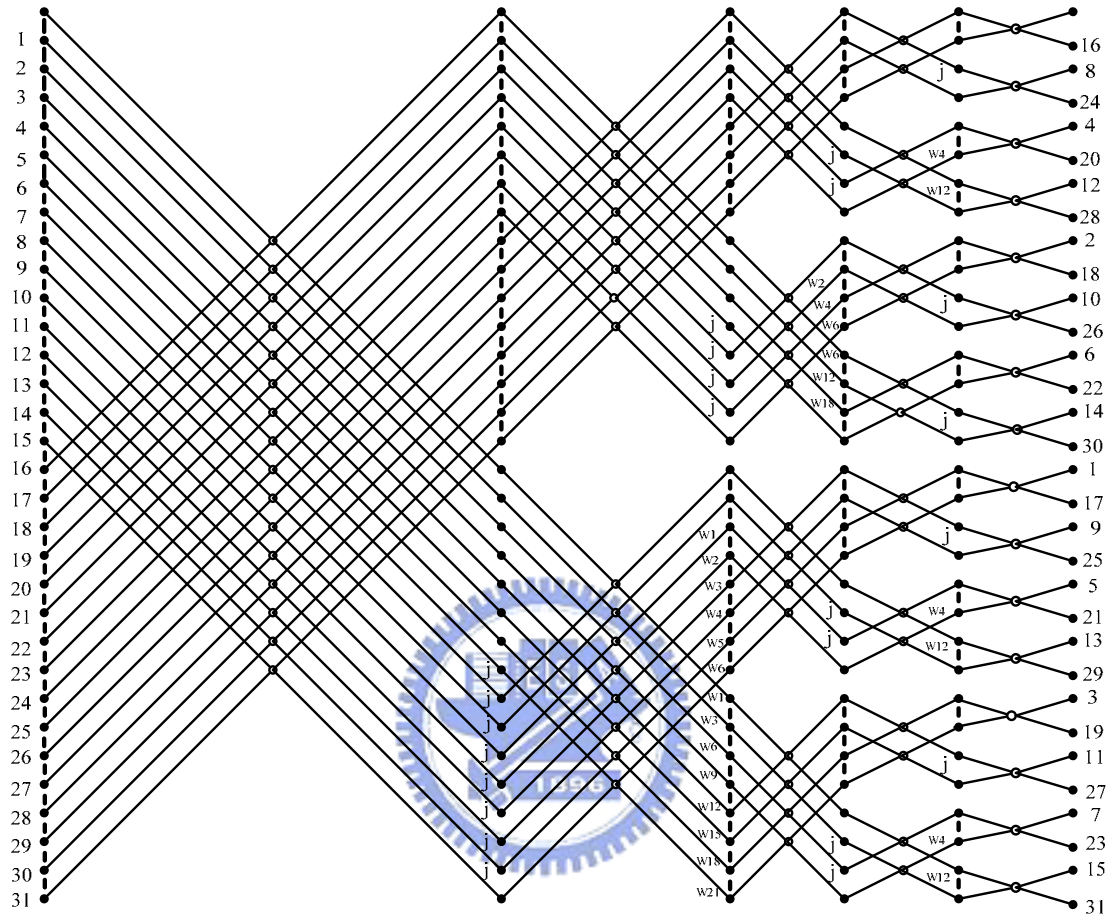


Fig 4.26 Flowchart of the length 32 Split-Radix FFT algorithm

#### 4.4.2 Evaluation and Comparison Results

Reminding the developmental roadmap discussed in the first part of Chapter 3, the ALU cluster IPs will be combined with the versatile baseboard to form a media streaming architecture with homogeneous processor cores. As shown in Fig 3.4, different numbers of ALU cluster IPs will be stacked with the board. As the result of the simulator, the numbers of the ALU cluster IPs are decided. One, two, four and eight ALU cluster IPs will be integrated with the baseboard in order. As introduced above, there are two target architectures to evaluate the performance and any of which will be considered when different numbers of ALU cluster IPs are integrated.

The two target architectures used in the thesis are Original Integer Architecture and Floating Point Unit and Original Integer Architecture Mixed. First, the original integer architecture is the architecture of the ALU cluster IP mentioned in Section 3.3.2. It is the architecture for integer operations essentially. The benchmark with floating point operations is decomposed. All operand are presented in the IEEE 754 floating point format and decomposed into several fields of integer and computes separately. It means the floating point operations are executed on the integer operation architecture by decomposing the format fit the original form. Then the second evaluation architecture is the floating point unit and original integer architecture mixed. In the SRFFT benchmark, it includes integer operations and floating point number operations. For the second evaluation architecture the integer operations and floating point number operations are computed by the integer and floating point unit respectively.

Before the evaluation results are presented, the essential cycles of executing the floating point numbers by original integer architecture such as ALU cluster IP will be calculated. The multiplication operation of the floating point numbers needs six cycles to finish the calculation. The addition operations of the floating point numbers in the ALU unit costs seven cycles to complete the calculation. In addition to these floating point number operations, other integer operations of this architecture need one cycle to finish the operation. The first data inputted into the functional units will need more cycles to finish the job result from the pipeline features. The integer ALU operations of the ALU cluster IP is two-stage pipelined and the integer multiplication operations of the ALU cluster IP is four-stage pipelined. The floating point number operations executed in the floating point unit architecture need one cycle for finishing the addition and one cycle for completing the multiplication. In the performance evaluation in the floating point unit architecture, all of the data are represented by the IEEE 754 format and calculate in this architecture.

Different number of clusters will be considered and evaluated with the two target architectures. The essential cycles to complete the 32-points FFT benchmark between different number clusters and different target architectures are listed separately in the following tables. They show the cycles need to finish the benchmark. The leftmost field stands for the ALU cluster IPs included in the evaluation. The middle field listed in the table represents the operation cycles needed for the functional unit while executing the benchmark. The rightmost field of the tables is the critical cycles which is underlined and dominate the performance of the conditions of different number clusters and different target architectures.

Table 4.6 shows the detail information of cycles for the Original Integer Architecture, the ALU cluster IP which is mentioned in former section. When only one cluster is used in the evaluation, the dominant functional unit is ALU2 and the dominant cycles of the performance is 647 cycles. The same concepts as one cluster, the evaluation for two clusters insides are listed and the dominant functional unit is ALU1 in the second cluster and the dominant cycles is 331 cycles. As discussed just now, the performance of four clusters insides architecture is that the dominant functional unit is ALU2 in the second cluster and the dominant cycles are 213 cycles. The dominant functional unit is ALU1 of the fifth cluster in the eight clusters architecture and its cycles are 151 cycles.

Table 4.6 Performance Evaluation Results for Original Integer Architecture

	Function Unit and Cycles	Critical Cycles
1 Cluster	ALU1:645 ALU2:647 MUL1:412 MUL2:330	<b><u>647</u></b>
2 Clusters	ALU1:320 ALU2:319 MUL1:141 MUL2:95	320
	ALU1:331 ALU2:309 MUL1:256 MUL2:240	<b><u>331</u></b>
4 Clusters	ALU1:177 ALU2:213 MUL1:55 MUL2:72	<b><u>213</u></b>
	ALU1:190 ALU2:153 MUL1:105 MUL2:72	190
	ALU1:173 ALU2:179 MUL1:135 MUL2:112	179
	ALU1:154 ALU2:141 MUL1:189 MUL2:122	189
8 Clusters	ALU1:145 ALU2:96 MUL1:39 MUL2:34	145
	ALU1:96 ALU2:126 MUL1:52 MUL2:34	126
	ALU1:85 ALU2:90 MUL1:54 MUL2:54	90
	ALU1:91 ALU2:60 MUL1:67 MUL2:44	91
	ALU1:151 ALU2:132 MUL1:89 MUL2:84	<b><u>151</u></b>
	ALU1:66 ALU2:103 MUL1:92 MUL2:64	103
	ALU1:97 ALU2:90 MUL1:74 MUL2:74	97
	ALU1:72 ALU2:102 MUL1:97 MUL2:64	102

Table 4.7 shows the detail information of cycles for the Floating Point Unit and Original Integer Architecture Mixed. It processes the integer numbers and the floating point numbers separately. When only one cluster is used in the evaluation, the dominant functional unit is ALU2 and the dominant cycles of the performance is 157 cycles. The same concepts as one cluster, the evaluation for two clusters insides are listed and the dominant functional unit is ALU2 both in the first cluster and in the second cluster and the dominant cycles is 74 cycles. As discussed just now, the performance of four clusters insides architecture is that the dominant functional unit is ALU1 in the third cluster and the dominant cycles are 52 cycles. The dominant functional unit is ALU1 of the fifth cluster and ALU2 of the sixth cluster in the eight clusters architecture and its cycles are 42 cycles.

Table 4.7 Performance Evaluation Results for Floating Point Unit and Original Integer Architecture Mixed

	Function Unit and Cycles	Critical Cycles
1 Cluster	ALU1:150 ALU2:154 MUL1:157 MUL2:143	<u>157</u>
2 Clusters	ALU1:71 ALU2:74 MUL1:74 MUL2:68	<u>74</u>
	ALU1:69 ALU2:74 MUL1:73 MUL2:67	<u>74</u>
4 Clusters	ALU1:48 ALU2:48 MUL1:46 MUL2:42	48
	ALU1:50 ALU2:50 MUL1:48 MUL2:43	50
	ALU1:52 ALU2:50 MUL1:47 MUL2:43	<u>52</u>
	ALU1:50 ALU2:49 MUL1:48 MUL2:44	50
8 Clusters	ALU1:40 ALU2:39 MUL1:35 MUL2:34	40
	ALU1:38 ALU2:41 MUL1:39 MUL2:34	41
	ALU1:40 ALU2:40 MUL1:35 MUL2:35	40
	ALU1:40 ALU2:38 MUL1:40 MUL2:35	40
	ALU1:42 ALU2:38 MUL1:38 MUL2:36	<u>42</u>
	ALU1:38 ALU2:42 MUL1:42 MUL2:36	<u>42</u>
	ALU1:40 ALU2:40 MUL1:36 MUL2:36	40
	ALU1:40 ALU2:39 MUL1:40 MUL2:36	40



As listed and described above, two target architectures with different numbers of clusters are evaluated. Considering the performance of the information listed above. In order to compare the performance of the selected benchmark, the variable of the number of clusters inside will be fixed between these architectures. First, the performance when one cluster inside these architectures are compared and plotted in Fig 4.27. The cycles used to complete the benchmark is 647 and 157 cycles for the original integer architecture and the mixture of floating point unit and original integer architecture respectively. Result from analyzing the data above, the performance of the mixed floating point unit and original integer architecture is 4.12 times better than the original integer architecture in cycles.

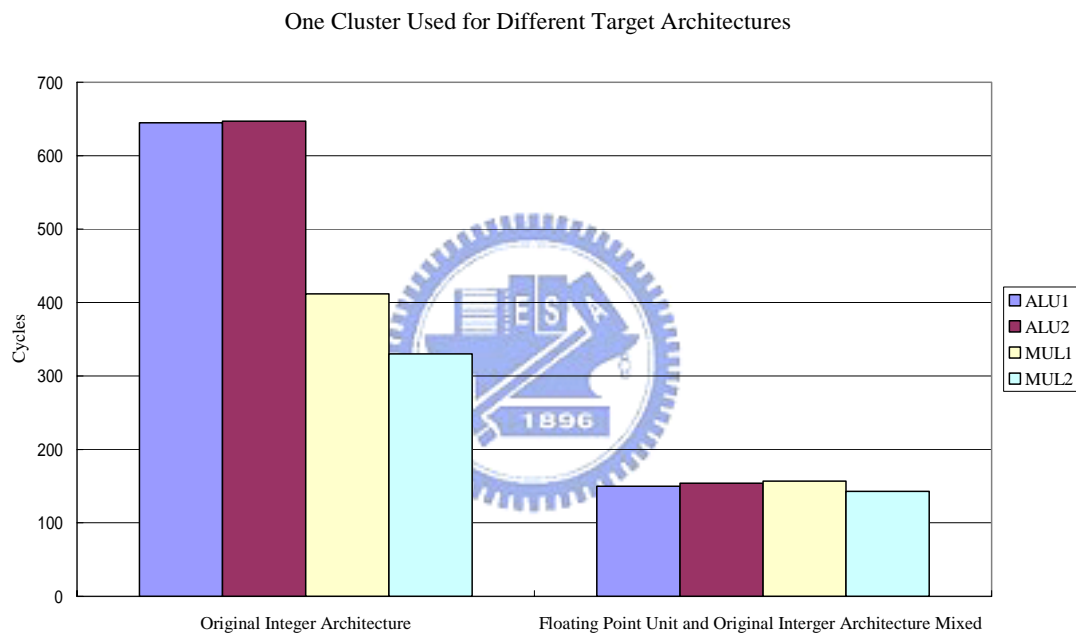


Fig 4.27 Performance Evaluation of one cluster included in these architectures

Second, the performance when two clusters included in these architectures are compared and plotted in Fig 4.28. The essential cycles used to complete the benchmark is 331 and 74 cycles for the original integer architecture and floating point unit and original integer architecture mixed respectively. Result from analyzing the data above, the performance of the mixed of floating point unit and original integer architecture is 4.47 times better than the original integer architecture in cycles.

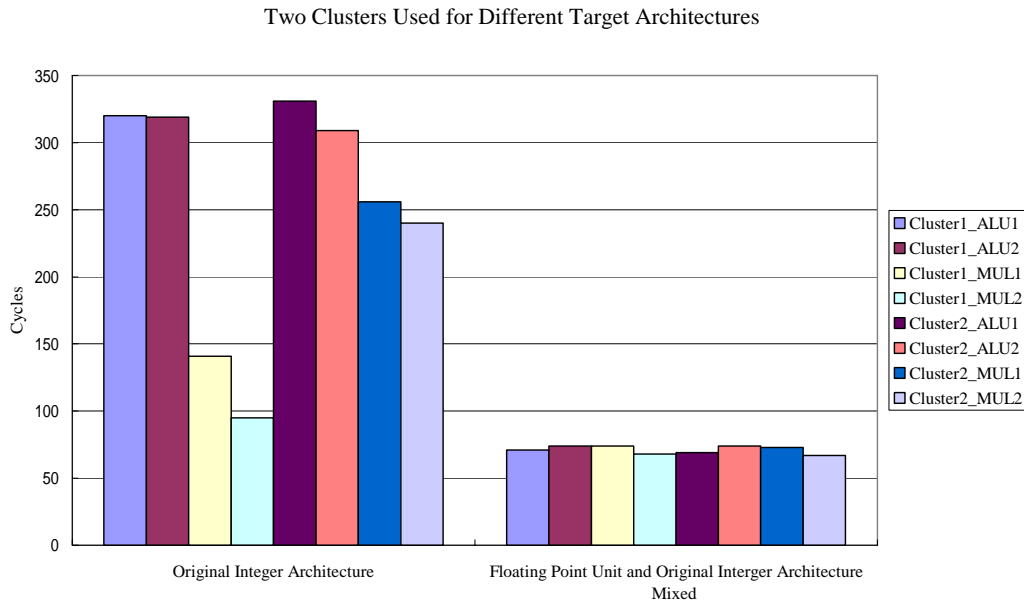


Fig 4.28 Performance Evaluation of two clusters included in these architectures

Following that the performance when four clusters included in these architectures are compared and sketched in Fig 4.29. The critical cycles need to finish the benchmark is 213 and 52 cycles for the original integer architecture and the floating point unit and original integer architecture respectively. Analyzing the data above the performance of the mixture of floating point unit and original integer architecture is 4.09 times better than the original integer architecture in cycles.

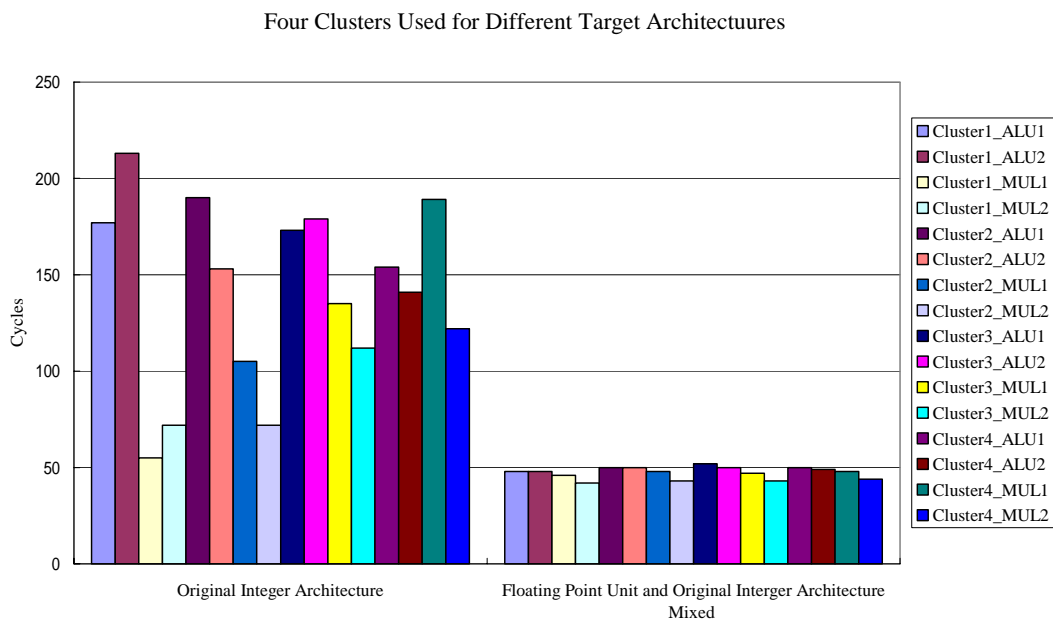


Fig 4.29 Performance Evaluation of four clusters included in these architectures

Finally the performance when eight clusters included in these architectures are compared and sketched in Fig 4.30. The essential cycles need to finish the benchmark is 151 and 42 cycles for the original integer architecture and mixed floating point unit and original integer architecture respectively. Analyzing the data above the performance of the mixed floating point unit and original integer architecture is 3.60 times better than the original integer architecture in cycles.

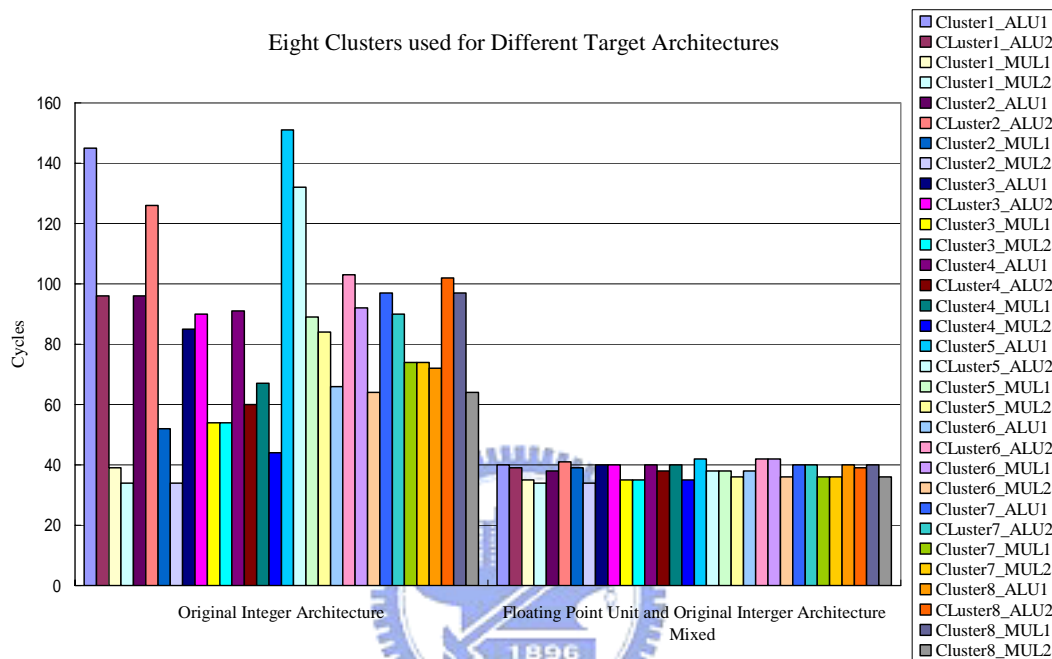


Fig 4.30 Performance Evaluation of eight clusters included in these architectures

Because of the hardware implementation, these architectures may operate in different clock rate. As described in the previous sections, the original integer architecture, the ALU cluster IP, is operated in the clock frequency of 100 MHz for post-layout simulation. The floating point unit adopted in the floating point unit and original integer architecture mixed are designed and implemented with the 75 MHz clock rate in the post-layout simulation. So the comparison of the execution time between these architectures is evaluated also. As listed in the Table 4.8, the critical execution time of the benchmark for two target architectures with different numbers of clusters is shown. The boldface and underlined data are listed to represent the critical execution time needed for completing the benchmark. Result from scheduling the instructions of the operations in the SRFFT benchmark the critical functional units in these architectures is slightly different. When scheduling the instructions, the cycles which execute no-operation (NOP) instructions will influence the total execution time

result in this situation. The results will be compared each other later to demonstrate the trend of the performance evaluation is in similar with the performance evaluation in clock cycles.

Table 4.8 Performance Evaluation Results in Execution Time

Execution Time (ns)	Original Integer Architecture	Floating Point Unit and Original Integer Architecture Mixed
1 Cluster	<b><u>6470</u></b>	<b><u>2035.3</u></b>
2 Cluster	3200	<b><u>957.8</u></b>
	<b><u>3310</u></b>	944.5
4 Cluster	<b><u>2130</u></b>	602.1
	1900	622.1
	1790	<b><u>628.9</u></b>
	1890	625.2
8 Cluster	1450	505.6
	1260	525.5
	900	518.8
	910	518.8
	<b><u>1510</u></b>	532.2
	1030	<b><u>545.4</u></b>
	970	518.8
	102	518.8

As in Table 4.8 and detail information of the Fig 4.31 shown below, the performance when one cluster inside between these architectures are compared and plotted in Fig 4.31. The essential execution time needed to complete the benchmark is 6470 and 2035.3 nano seconds for the original integer architecture and mixed

architecture of floating point unit and original integer architecture respectively. Analyzing the data above, the performance of the mixture of floating point unit and original integer architecture is 3.18 times better than the original integer architecture.

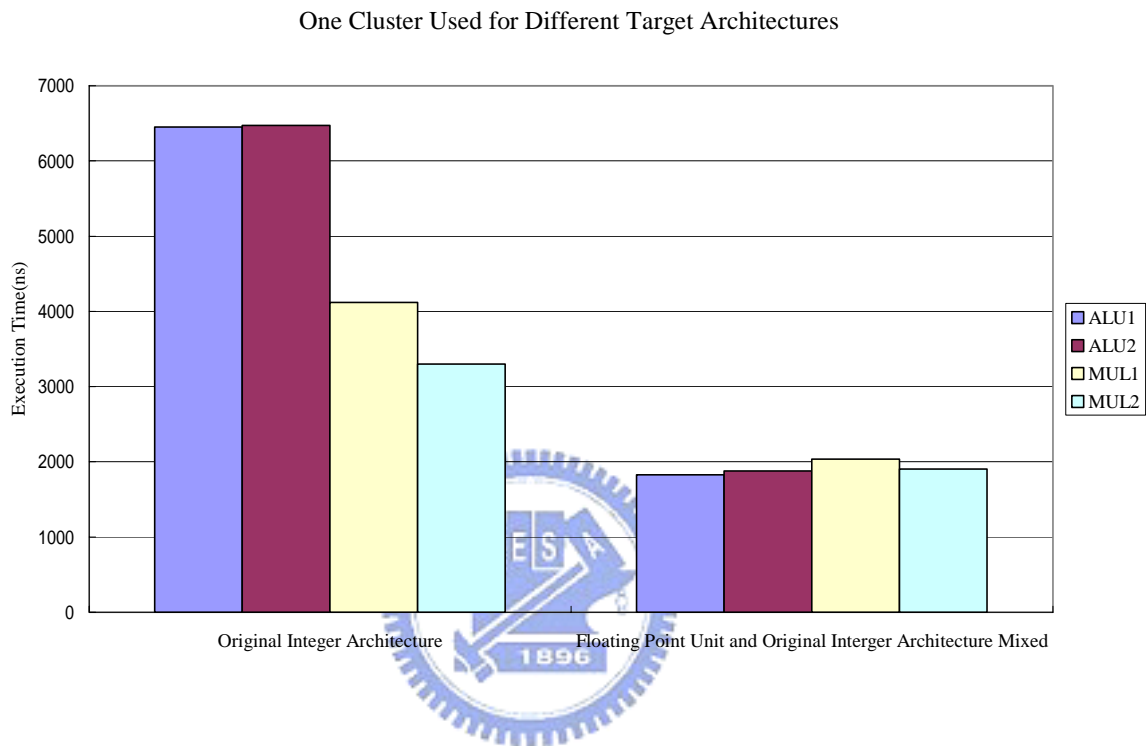


Fig 4.31 Performance Evaluation of one cluster included in execution time

Second, the detail information of the performance when two clusters included in these architectures are compared and plotted in Fig 4.32 and Table 4.8. The needed execution time used to complete the benchmark is 3310 and 957.8 nano seconds for the original integer architecture and the mixture of floating point unit and original integer architecture respectively. Result from analyzing the data above, the performance of the mixed of floating point unit and original integer architecture is 3.46 times better than the original integer architecture.

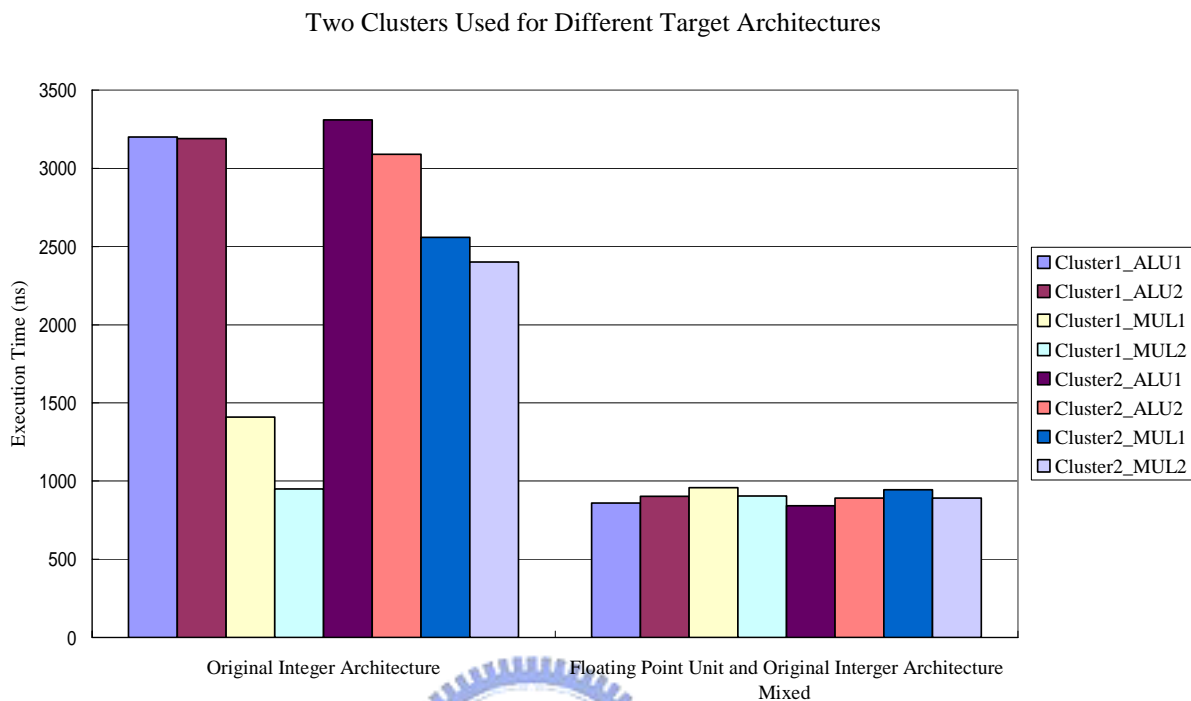


Fig 4.32 Performance Evaluation of two clusters included in execution time

Third, the detail information of the performance when four clusters included in these architectures are compared and plotted in Fig 4.33 and Table 4.8. The critical time need to finish the benchmark is 2130 and 628.9 nano seconds for the original integer architecture and the mixture of floating point unit and original integer architecture respectively. Analyzing the data above the performance of the mixed floating point unit and original integer architecture is about 3.39 times better than the original integer architecture in seconds.

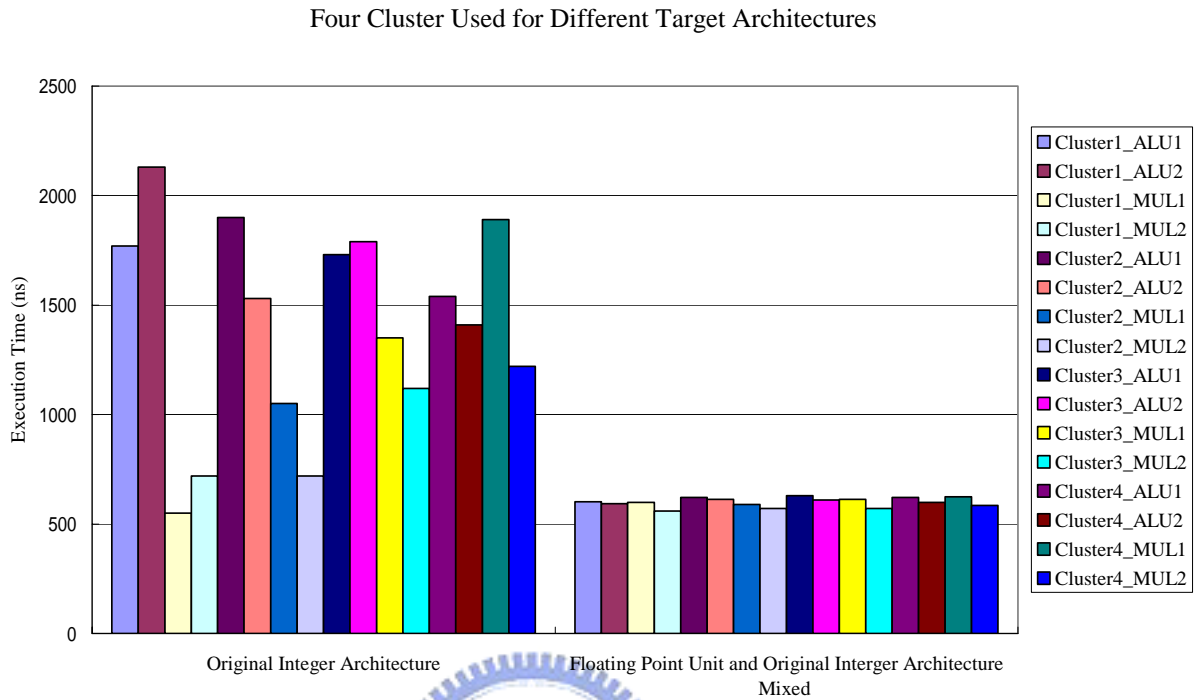


Fig 4.33 Performance Evaluation of four clusters included in execution time

Eventually the performance when eight clusters included in these architectures are compared and sketched in Fig 4.34. The essential time need to complete the benchmark is 1510 and 545.4 nano seconds for the original integer architecture and floating point unit and original integer mixed architecture respectively. Result from analyzing the data above, the performance of the mixture of floating point unit and original integer architecture is about 2.77 times better than the original integer architecture.

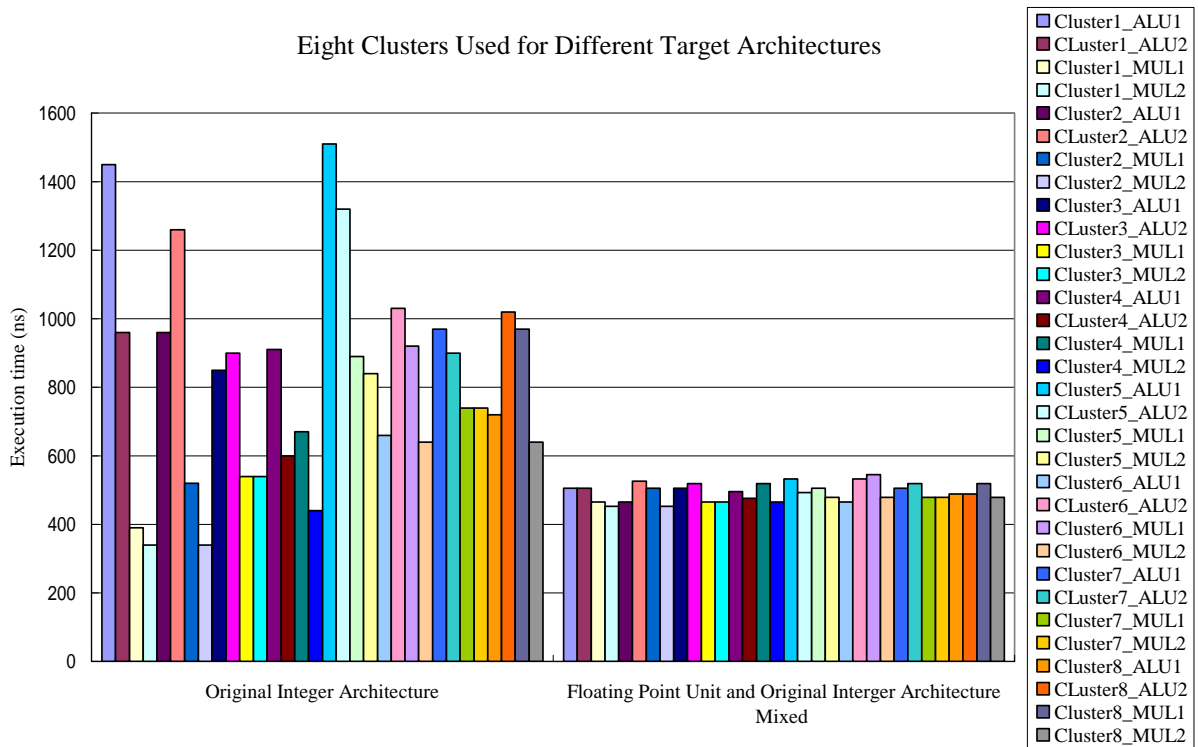


Fig 4.34 Performance Evaluation of eight clusters included in execution time

The summary of the descriptions above are discussed in this paragraph. The performance evaluations considering the execution cycles and the execution time of two target architectures with different numbers of clusters used are normalized to the original integer architecture such as the architecture of the ALU cluster IP mentioned in previous section and the amount of performance enhancement are listed. As shown in Fig 4.35, the performance normalized to the original integer architecture is sketched. Observing Fig 4.35, it is clear that no matter what numbers of clusters included the trend of the performance increases incrementally. Considering the phenomenon that more clusters used makes the performance improvement increases slight slowly when the number of clusters used is focused between different target architectures. This phenomenon results form that more cluster used will share and degrade the computation loading of each functional unit and reveals the slight slowly increase of performance. In spite of this, it is still about four times of the performance improvement when the floating point units involved in the architectures.

The performance evaluation considering the execution time has the same trend and phenomenon. As illustrated in Fig 4.36, the trend of the performance increases incrementally regardless of the numbers of clusters. It also has the phenomenon that the performance increases slight slowly when more clusters used if the number of



clusters used is focused between different target architectures. The reason is the same as description above. In spite of this, it is still about 3.3 times of the performance improvement when the floating point units involved in the architectures. These two performance evaluation executed the benchmark SRFFT prove that the floating point unit is an essential units to improve the performance of the architecture.

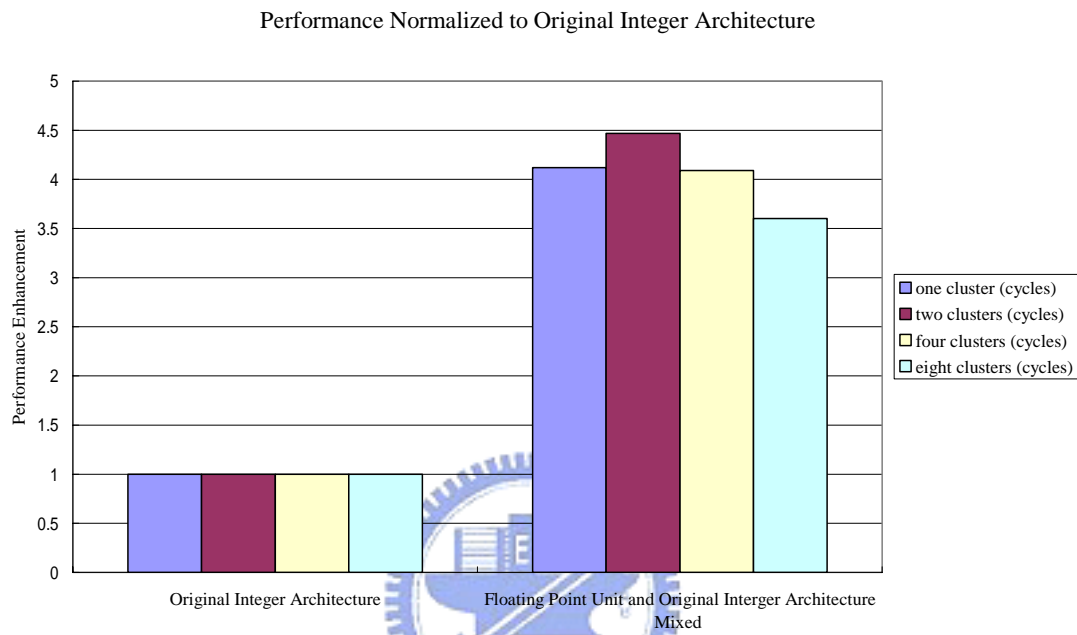


Fig 4.35 Comparison of Performance Normalized in execution cycles

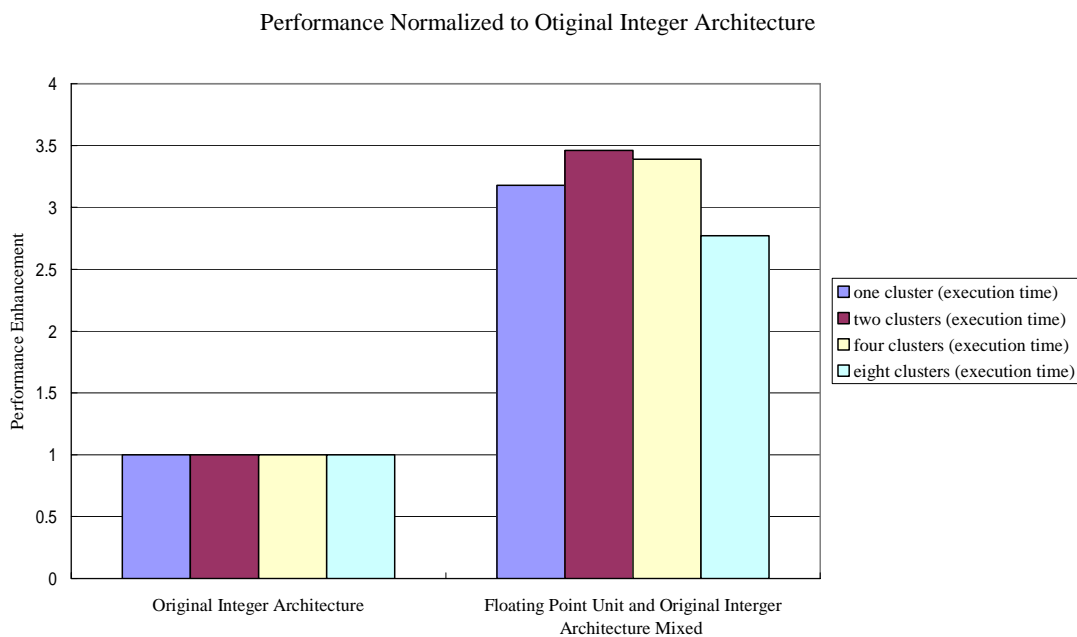


Fig 4.36 Comparison of Performance Normalized in execution time

The following of this section the performance comparison results which fixed the type of adopted architecture when different numbers of clusters included in the selected architecture are presented. Consider the execution cycles of finishing the SRFFT benchmark, the results are shown in Fig 4.37. As illustrated in the figure, the trend shows that the more cluster used in the architecture the higher performance which is gained from the architecture. The performance enhancements of the original integer architecture are 1.95, 3.04 and 4.28 times higher than one cluster included architecture for two clusters, four clusters and eight clusters used in the original integer architecture respectively. Similarly the mixture architecture of the floating point unit and original integer architecture also has the trend. The performance improvements of the selected mixture architecture are 2.12, 3.01 and 3.73 times higher than single cluster used for two clusters, four clusters and eight clusters used respectively.

Then the performance comparisons in the perspective of execution time are also presented. As shown in Fig 4.38, the trend of performance improvement is the same. As illustrated in the figure, the performance enhancements of the original integer architecture are 1.95, 3.03 and 4.28 times higher than one cluster included architecture for two clusters, four clusters and eight clusters used in the original integer architecture respectively. Similarly the mixture architecture of the floating point unit and original integer architecture also has the trend in the perspective of execution time. The performance improvements of the selected mixture architecture are 2.12, 3.24 and 3.73 times higher than single cluster used for two clusters, four clusters and eight clusters used respectively. The results mentioned above shows that more clusters used in the selected architecture makes more parallelism be exploit and improves the performance higher.

A phenomenon observed from Fig 4.37 and Fig 4.38 is discussed in this paragraph. It shows that when the numbers of clusters included in the architecture are increased from one to two or from two to four, the performance improvement is doubled no matter what architectures are adopted and either in the perspective of execution time or execution cycles. But when the clusters included increase from four to eight, the performance enhancement is not doubled and not conspicuous. The phenomenon results from huge data exchange outside the intra register files embedded in each arithmetic unit and shows that the performance enhancement does not come with more and expensive hardware resources in this case.

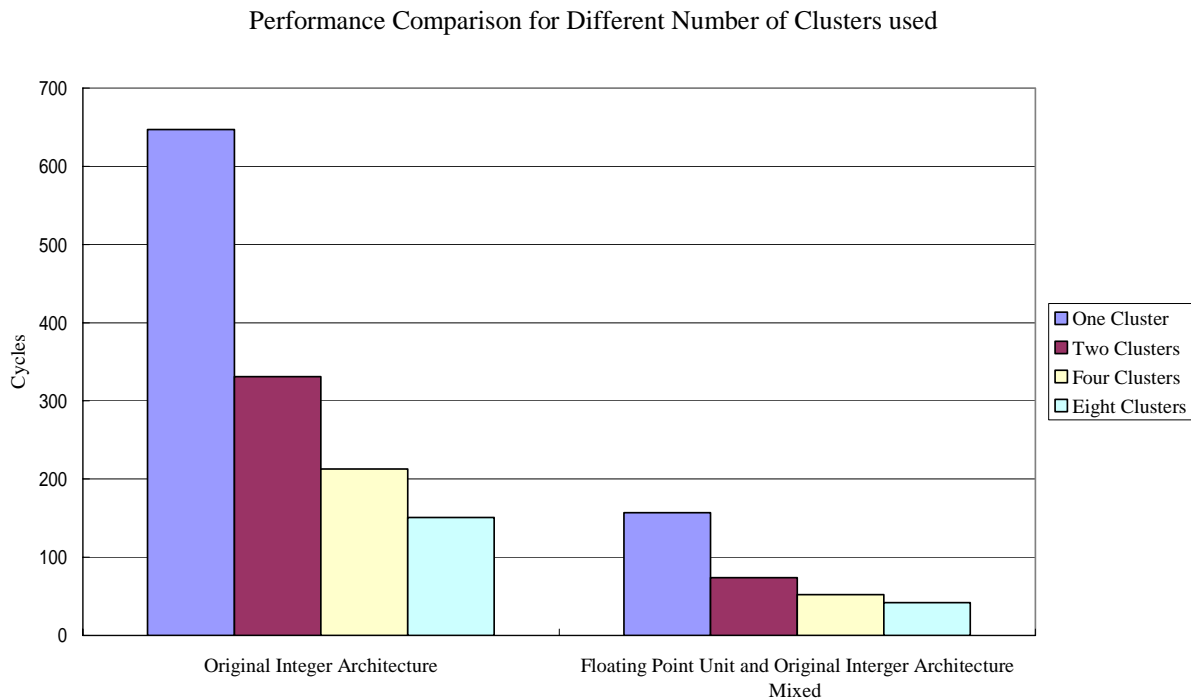


Fig 4.37 Performance Comparison for Different Number of Clusters used in cycles

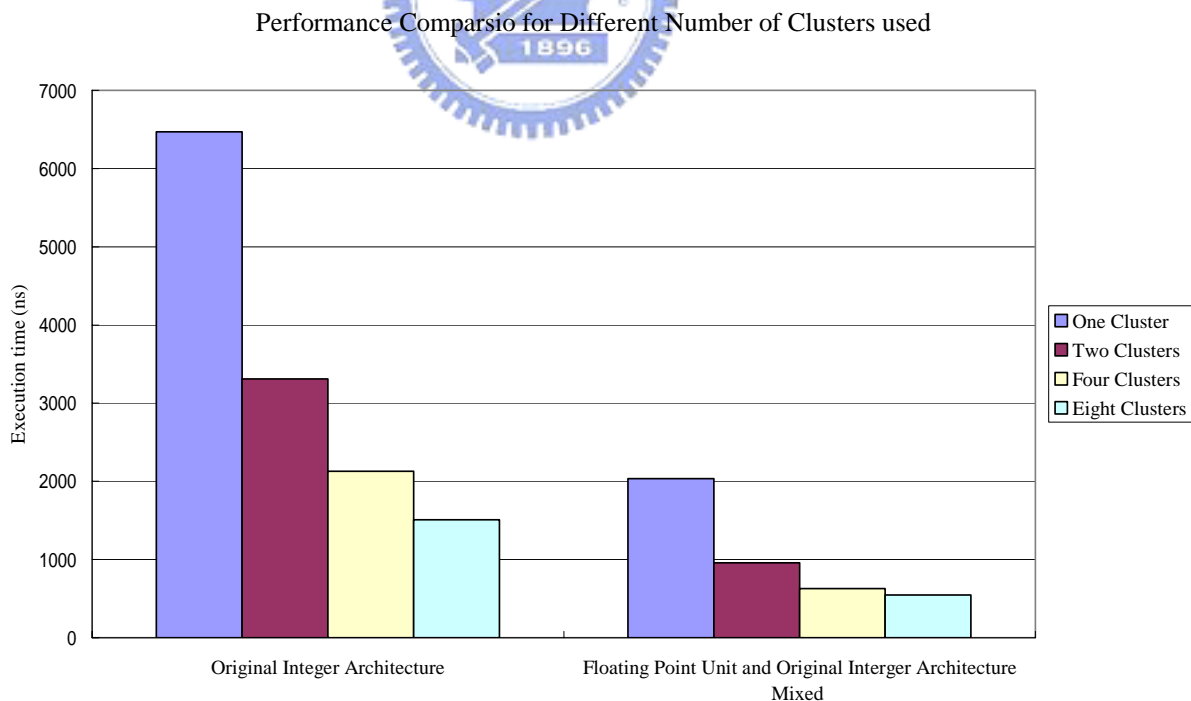


Fig 4.38 Performance Comparison for Different Number of Clusters used in execution time

The summarized conclusions of the section of the performance evaluation are described in this paragraph. The performance enhancement when the floating point unit involved in the target architecture is 4.07 times in the condition of execution cycles. In the condition of execution time it also provides 3.2 times of performance improvement in the discussed architecture. In the analysis and evaluation of this thesis, more clusters used in each architecture result from higher performance and the most performance enhancement is 4 times while eight clusters adopted. The area overhead while providing the performance improvement discussed above is 10.8% by the floating point unit.



# CHAPTER 5

## Conclusion and Future Work

In this chapter, the conclusions of proposed designs in this thesis are described. The future work of constructing the media processing architecture with homogeneous processor cores is also discussed. The details are introduced in the following sub-sections.

### 5.1 Conclusion

In this thesis, the media processing architecture with homogeneous processor cores is proposed to overcome the challenges of the programming models and system architectures. In the meantime, this work has also demonstrated the feasibility of implementation consideration. An ALU cluster IP designed shows it is suitable for media application and such a hardware accelerator with platform-based design is able to form a media streaming architecture with homogeneous processor cores in the future. The chip is manufactured using TSMC 0.15um CMOS technology. The details of the silicon backed design are addressed in the previous chapters.

The proposed ALU cluster IP with Magnetic RAM provides the ability to against the issue of soft error [35 - 37]. It through the multiple functional units and replace the data memory with Magnetic RAM to form a radiation harden architecture. It will be a critical issue of the highly integrated design and the design for typical and special applications such as mission critical, life critical, military and space applications.

To go a step further, the ALU cluster IP which supported floating point operations is also proposed. The floating point units used to integrate with the ALU cluster IP are designed and implemented as hard macros. These hard macros of floating point units provide efficient processing ability to handle the operation of floating point numbers. The performance evaluation of variant architectures reveals

and proves the floating point units designed is essential and critical in the proposed hardware accelerator.

Continuing the design of the media streaming architecture with homogeneous processor cores there are future works needed to be implemented, designed and make effort. The details are introduced in the following paragraph.

## 5.2 Future Work

As shown in Fig 3.4, the prototype 3, the ALU cluster IP with floating point operation supported will be taped out. Then the system integration will preliminary start. An ALU cluster IP with compatible board for logic tile connector will be stacked into the RealView versatile platform baseboard for ARM926EJ-S shown in Fig 5.1. Besides verifying the AMBA protocol of chip, one ALU cluster IP stacked in the board also shows that the IP has the capability as a hardware accelerator integrated with the platform.



Fig 5.1 RealView Versatile Platform Baseboard for ARM926EJ-S

Subsequently multiple processing elements, ALU cluster IPs, will be integrated with the versatile baseboard to form a media streaming architecture with homogeneous processor cores. Suitable and selected benchmarks are ported into this processing system and compare with each other. Eventually a high efficient, well matched to media application and radiation harden processing system will be proposed and completed.

## BIBLIOGRAPHY

- [1] Rixner, S., Dally, W.J., Kapasi, U.J., Khailany, B., Lopez-Lagunas, A., Mattson, P.R., Owens, J.D.,” A bandwidth-efficient architecture for media processing,” *Proceedings. 31st Annual ACM/IEEE International Symposium on Microarchitecture*, Pages:3 – 13, 30 Nov.-2 Dec. 1998.
- [2] W. J. Dally, U. J. Kapasi, B. Khailany, J. H. Ahn, A. Das, “Stream Processors: Programmability with Efficiency,” *ACM Queue*, pages 52-62, March 2004.
- [3] B. Khailany, J. W. Dally, U. J. Kapasi, P. Mattson, J. Namkoong, J. D. Owens, B. Towles, A. Chang, S. Rixner, “Imagine: Media Processing with Streams,” *IEEE Micro*, pages 35-46, March-April 2001.
- [4] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, J. D. Owens, “Programmable Stream Processors,” *IEEE Computer*, pages 54-62, August 2003.
- [5] K. Mai, T. Paaske, N. Jayasena, R. Ho, J. W. Dally, M. Horowitz, “Smart Memories: A Modular Reconfigurable Architecture,” *Proceedings of the 27th International Symposium on Computer Architecture*, pages 161-171, June 2000.
- [6] Owens, J.D., Rixner, S., Kapasi, U.J., Mattson, P., Towles, B., Serebrin, B., Dally, W.J.,” Media processing applications on the Imagine stream processor,” *Proceedings of the 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'02)*, Pages:295 – 302, Sept. 2002.
- [7] Jung Ho Ahn, Dally, W.J., Khailany, B., Kapasi, U.J., Das, A,” Evaluating the Imagine Stream Architecture,” *Proceedings of 31st Annual International Symposium on Computer Architecture*, Pages: 14 – 25, June. 2004.
- [8] L. Hennessy, A. Patterson, *Computer Architecture: A Quantitative Approach*, Third Edition, Morgan Kaufmann Publishers, 2003.
- [9] W. Wolf, *Modern VLSI Design: System-on-Chip Design*, Third Edition, Prentice Hall Modern Semiconductor Design Series, 2002.
- [10] Neil. H. E. Weste, David Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, Third Edition, Addison-Wesley VLSI Systems Series, 2005.

- [11] J. Rabaey, A. Chandrakasan, B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, Second Edition, Prentice Hall Electronics and VLSI Series, 2003.
- [12] Nsame, P., Savaria, Y.,” A Customizable Embedded SoC Platform Architecture,” *Proceedings of 4th IEEE International Workshop on System-on-Chip for Real-Time Applications*, Pages:299 - 304, July 2004.
- [13] Ghattas, H., Mbaye, M., Pepga Bissou, J., Savaria, Y.,” SoC Platform Architecture for a Network Processor,” *Proceedings of International Symposium on System-on-Chip*, Pages:49 - 52, Nov. 2003.
- [14] Sangiovanni-Vincentelli, A., Martin, G.,” Platform-based design and software design methodology for embedded systems,” *IEEE Design & Test of Computers*, Pages:23 - 33, Nov.-Dec. 2001.
- [15] Todman, T.J., Constantinides, G.A., Wilton, S.J.E.; Mencer, O., Luk, W.; Cheung, P.Y.K.,” Reconfigurable computing: architectures and design methods,” *IEE Proceedings-Computers and Digital Techniques*, Pages:193 - 207, Mar. 2005.
- [16] Singh, H., Ming-Hau Lee, Guangming Lu, Kurdahi, F.J., Bagherzadeh, N., Chaves Filho, E.M.,” MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications,” *IEEE Transactions on Computers*, Pages:465 - 481, May 2000.
- [17] Bocchi, M., de Dominicis, M., Mucci, C., Deledda, A., Campi, F., Lodi, A., Toma, M., Guerrieri, R.,” Design and implementation of a reconfigurable heterogeneous multiprocessor SoC,” *IEEE Custom Integrated Circuits Conference 2006*, Pages:93 - 96, Sept. 2006.
- [18] Kapasi, U.J., Dally, W.J., Rixner, S., Owens, J.D., Khailany, B.,” The Imagine Stream Processor,” *Proceedings of the 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'02)*, Pages:282 – 288, Sept. 2002.
- [19] Ting-Wei Lin, Ming-Chung Lee, Fang-Ju Lin, Herming Chiueh, “A Low Power ALU Cluster Design for Media Streaming Architecture,” *IEEE 48th International Midwest Symposium on Circuits and Systems*, Pages:51 - 54 Aug. 2005.
- [20] Ting-Wei Lin, “An ALU cluster Design for Media Streaming processors Architecture,” Master dissertation, University of Chiao Tung University, 2005.



- [21] ARM Corp., “RealView Platform Baseboard for ARM926EJ-S HBI-0117 User Guide,”  
[http://www.arm.com/pdfs/DUI0224C\\_pb926ej-s\\_user\\_guide.pdf](http://www.arm.com/pdfs/DUI0224C_pb926ej-s_user_guide.pdf)
- [22] ARM Corp., “Versatile/LT-XC2V4000+ Logic Tile User Guide,”  
[http://www.arm.com/pdfs/DUI0186B\\_lt\\_xc2v\\_userguide.pdf](http://www.arm.com/pdfs/DUI0186B_lt_xc2v_userguide.pdf)
- [23] Garrett, D., Davis, L., ten Brink, S., Hochwald, B., Knagge, G.,” Silicon complexity for maximum likelihood MIMO detection using spherical decoding,” *IEEE Journal of Solid-State Circuits*, Pages:1544 - 1552, Sept. 2004.
- [24] Shao-Hsuan Chang, “Design and Implementation of an ALU Cluster Intellectual Property as a Reconfigurable Hardware Accelerator for Media Streaming Architecture,” Master dissertation, University of Chiao Tung University, 2006.
- [25] ARM Corp.,” AMBA™ Specification (Rev 2.0)  
[http://www.arm.com/products/solutions/AMBA\\_Spec.html](http://www.arm.com/products/solutions/AMBA_Spec.html)
- [26] IEEE Computer Society (1985),” IEEE Standard for Binary Floating-Point Arithmetic,” *IEEE Std 754*, 1985
- [27] <http://www.agilent.com>
- [28] D.D. Tang, P.K. Wang, V. S. Speriosu, S. Le, R.E. Fontana, S. Rishton, “An IC process compatible Nonvolatile Magnetic RAM,” *Electron Devices Meeting*, pages 997-1000, 1995.
- [29] Durlam, M, Naji, P J, Omair, A, DeHerrera, M, Calder, J, Slaughter, J M, Engel, B N, Rizzo, N D, Grynkewich, G, Butcher, B, Tracy, C, Smith, K, Kyler, K W, Ren, J J, Molla, J A, Feil, W A, Williams, R G, Tehrani, S,” A 1-Mbit MRAM based on 1T1MTJ bit cell integrated with copper interconnects,” *IEEE Journal of Solid-State Circuits*, Pages: 769-773, May 2003.
- [30] A. V. Oppenheim, R. W. Schaffer, J. R. Buck, *Discrete-Time Signal Processing*, Second Edition, Prentice Hall Signal Processing Series, 1999.
- [31] Sorensen, H., Heideman, M., Burrus, C.,” On computing the split-radix FFT,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Pages:152 - 156, Feb. 1986.
- [32] Duhamel, P.,” Implementation of "Split-radix" FFT algorithms for complex, real, and real-symmetric data,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Pages:285 - 295, Apr. 1986.

- [33] User Guide and Reference Manual of Synopsys Online Document
- [34] Synopsys SolvNet  
<http://www.synopsys.com/support/support.html>
- [35] Saggese, G.P., Wang, N.J., Kalbarczyk, Z.T., Patel, S.J., Iyer, R.K.,” An experimental study of soft errors in microprocessors,” *IEEE Micro*, Pages: 30 - 39, Nov.-Dec. 2005.
- [36] Baumann, R.,” Soft errors in advanced computer systems,” *IEEE Design & Test of Computers*, Pages:258 - 266, May-June 2005.
- [37] Mukherjee, S.S., Emer, J., Reinhardt, S.K.,” The soft error problem: an architectural perspective,” *11th International Symposium on High-Performance Computer Architecture, 2005. HPCA-11.*, Page(s):243 - 247, Feb. 2005.

