國 立 交 通 大 學

電信工程學系

碩 士 論 文

多媒體串流處理器之運算單元設計

An ALU Cluster Design for Media Streaming Processors Architecture

研究生：林 庭 瑋

指導教授：闕 河 鳴 博士

中 華 民 國 九十四 年 九 月

多媒體串流處理器之運算單元設計

An ALU Cluster Design for Media Streaming Processors Architecture

研 究 生：林庭瑋　　　　Student：Ting-Wei Lin

指導教授：闕河鳴　博士　　　　Advisor：Dr. Herming Chiueh

國立交通大學

電信工程學系碩士班

碩士論文

A Thesis
Submitted to Department of Communication Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in
Communication Engineering
September 2005
Hsinchu, Taiwan.

中華民國九十四年九月

多媒體串流處理器之運算單元設計

研究生：林庭瑋　　　　　　　　　　　　　　　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

近期相關的研究已提出 利用串流架構來解決多媒體應用在傳統處理器架構上執行的缺點。除此之外，對行動式的系統而言，低功率消耗的考量已變成一個重要的議題，但是目前的串流架構並沒有針對以上的要求去解決。因此，在本研究中提出了將一個多媒體串流架構與數個低功率電路設計的方法結合在一起 來解決以上的要求。而本論文的目的是設計一個多媒體串流處理器的運算單元，且基於史丹佛大學所提出的串流處理器架構，在本研究的微架構上考慮了設計的可行性，及利用電腦輔助軟體的模擬結果來決定各個功能單元的架構。並且此架構結合了階層式的頻寬設計，來有效的利用記憶體之間的頻寬。在實驗數據的預估顯示上，針對多媒體應用及基頻通訊系統方面去執行所選定的標準檢查程式，並藉由動態的選擇所要使用的運算單元總數目，可將功率損耗與能量消耗變成可調整式的。因此，對於一個行動式的系統，其即時的效能表現與能量消耗之間的取捨成為可以做最佳化的。所以此提出的架構與相關類似的架構相比之下，已經對有限的電池壽命在功率消耗和執行時間之間提供一個重要的突破。

An ALU Cluster Design for
Media Streaming Processors Architecture

Student: Ting-Wei Lin Advisor: Dr. Herming Chiueh

Department of Communication Engineering

National Chiao Tung University

Hsinchu, Taiwan

Abstract

Recent research has proposed using streaming architecture to provide a leap in media applications that are poorly matched to conventional processor architecture. Besides, low power considerations are becoming an important issue for mobile systems, but streaming architecture solutions do not fit in above requirements. Therefore, in this research, a combination of media streaming architecture and low power circuitry design methodology is proposed. An ALU Cluster design for media streaming architecture is presented in this thesis, which is based on Stanford Imagine stream architecture with the consideration of implementation feasibility. The back-end simulation results decide the final micro-architecture of each component, and utilize communication bandwidth hierarchy design to effectively solve the problem of scarce memory bandwidth. The experimental results show that the power and energy consumption of selected benchmark for multimedia and baseband communication systems become scalable by dynamic selecting the number of utilized ALU Clusters. Thus, the instant performance and energy consumption of an entire work can be optimized for mobile systems. The proposed design has provided a breakthrough for similar architectures.

ACKNOWLEDGMENTS

This thesis would not have been possible without the support of many exceptional people. First and foremost, thanks go to my research advisor, Professor Herming Chiueh. He has always been an inspiration to me and everyone else on this project through his vision and leadership. He also provided irreplaceable guidance for me when I needed for a fascinating problem, good advice, constructive criticism, support, and flexibility.

I would also like to thank all team members of the SoC LAB group, especially my classmates over the years: Elliott Lee, Gary Chan, Maggie Lin, and Uan Cheng. They not only put up with me all of those years, but also made my days as an enjoyable graduate student.

Finally, I can not say enough about the support provided by my family. My parents have been my biggest supporters and for that I am forever grateful. My sister and girlfriend have always providing timely encouragement and advice. To all of my friends and family members who have helped me in one way or another over the years, I would like to say thanks.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Media processing applications have possession of three key important characteristics: large available parallelism, little data reuse, and a high computation of memory access ratio [1]. However, these characteristics are poorly matched to conventional general-purpose architectures. In the mean time, there is a processor-memory performance gap as well as a memory wall problem arisen that goes off-chip from processor to memory incurs severe latency and bandwidth penalties, as shown in Figure 1.1 [2]. In recent years, the current research has proposed to use the streaming architecture by fitting modern very large scale integrated-circuit (VLSI) technology with lots of arithmetic logic units (ALUs) on a single chip and the hierarchical communication bandwidth design to provide a leap in media processing applications [3][4][5]. Relative topics of recent research are Imagine Stream Processor [6][7][8], Smart Memories [9], and Processing-In-Memory [10][11]. Nevertheless, in contemporary VLSI circuitry for mobile systems, such as handheld audio and video applications, low power considerations are becoming an important issue as battery life and geometry of mobile systems are limited [12]. The streaming architecture and processor-in-memory solutions do not fit in above requirements since it generally occupies a huge die size to trade for the data and processing parallelism. Thus, in recent developments, most of these architectures are focused on the super computing architectures rather than the media processing applications.

However, the streaming architecture has been suggested as an efficient architecture for both media processing applications and baseband architecture by using the software defined radio mechanism [13][14]. In order to design the next generation portable multimedia and communication systems, the power dissipation of such a system is an emergency issue. Therefore, a low-power ALU Cluster for the streaming architecture is proposed, which combines the software-defined mechanism and the modern low-power circuitry technique in the streaming architecture to provide a breakthrough in the operating time and power dissipation in the limited battery power.

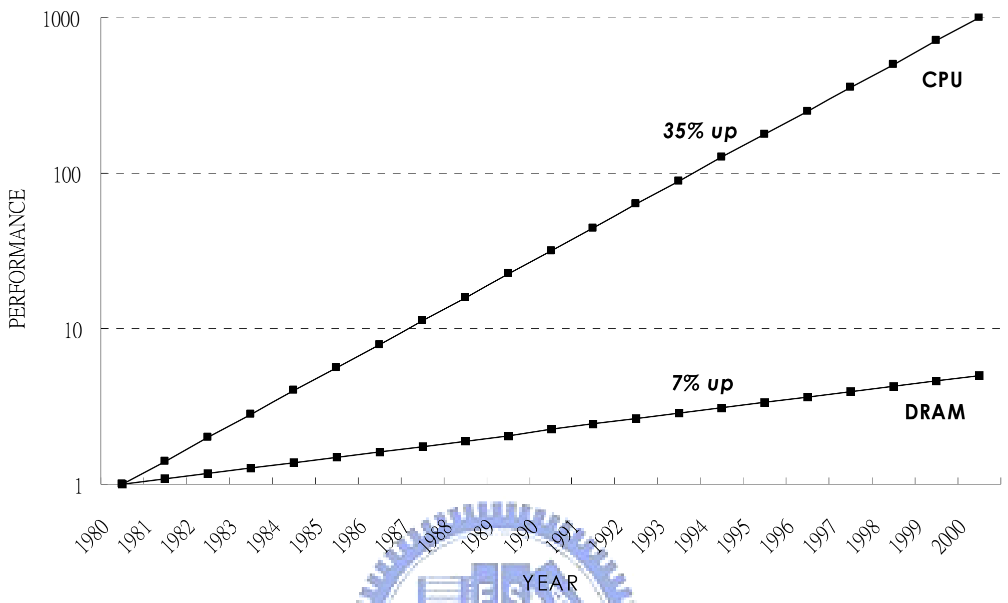


Figure 1.1: Gap between Processor and Memory

In this thesis, our major motivation is to improve media processing applications weakly matched to conventional processor architectures. In other words, we aim at the micro-architecture design of the 32-bit ALU Cluster for media streaming processor architectures because ALU Cluster is the nexus computation part of the processors and one main factor of increasing high kernel performance. However, two primary problems have been met are the required high computation throughput and the processor-memory performance gap. So our proposed solution methods to solve these performance bottlenecks are concurrency and locality, respectively. Concurrency is to provide abundant data-level parallelism which means moderate multiple function units in one ALU Cluster. Locality is to decrease the use of the global bandwidth to access the high latency off-chip memory, which means the temporary high speed storage units included inside the ALU Cluster that could form the memory bandwidth hierarchy architectures. With these solution methods, the performance of the media processing applications can be greatly improved.

This thesis focuses on an ALU Cluster architecture design of media streaming processors. The remaining of organization of this thesis is as follow.

In Chapter 2, the various design methodology styles, the streaming applications processing model, the current relative research topics, and the proposed media streaming processor architecture are introduced.

In Chapter 3, the detail micro-architecture of each components of ALU Cluster is designed. Instruction set format, and overall system pipeline operation from instructions read, data reads, operations execute to outcome writes back are also explained.

In Chapter 4, the electronic design automation (EDA) flow of implementing this work is presented. The benchmark simulation, performance evaluation, and comparison to recent related architecture design reports are discussed. Besides, a low power ALU Cluster design under group collaboration is also presented.

In Chapter 5, the conclusion of this proposed design is addressed.

CHAPTER 2
BACKGROUND

In this chapter, a briefly review of the background of the design methodology about three different design implementation styles, and three primary related research topics nowadays about the streaming architectures are described. In addition, the proposed media streaming processor architecture that bases on improving the disadvantages of the current existed streaming architectures is presented.

2.1 Design Methodology

Generally speaking, there are many different methods when implementing a design. In this section, three distinct design methodology styles which are application-specific architecture, platform-based architecture, and reconfigurable architecture will be briefly introduced on the basis of time to market demands, physical area, utilizing flexibility, etc. Furthermore, the pros and cons of these design methodology styles are also discussed.

First, the application-specific architecture design is easier among these architectures. Figure 2.1.1 is plotted an example of application-specific integrated circuit (ASIC) design. The chip implementation could be finished quickly as long as following the given well-defined specification. The physical area, operation frequency, and power dissipation could be optimized that depends on demands, too. Nevertheless, the application-specific architecture design is not so flexible and reusable, and needs to redo the overall design flow while the specific applications or specifications are changed.

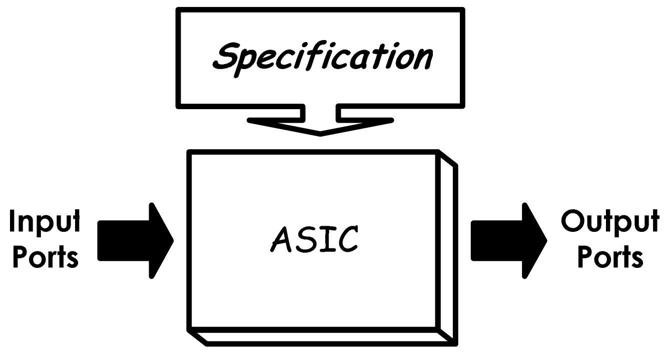
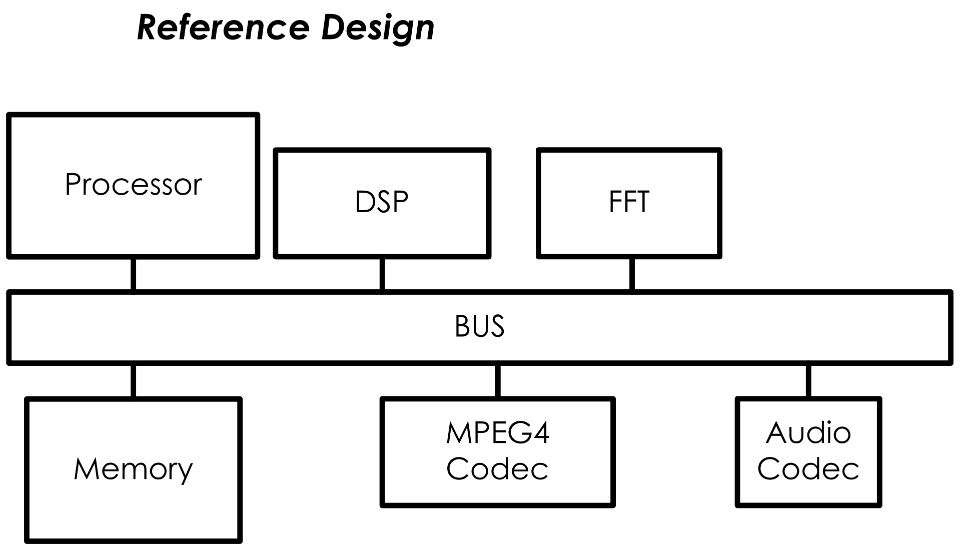
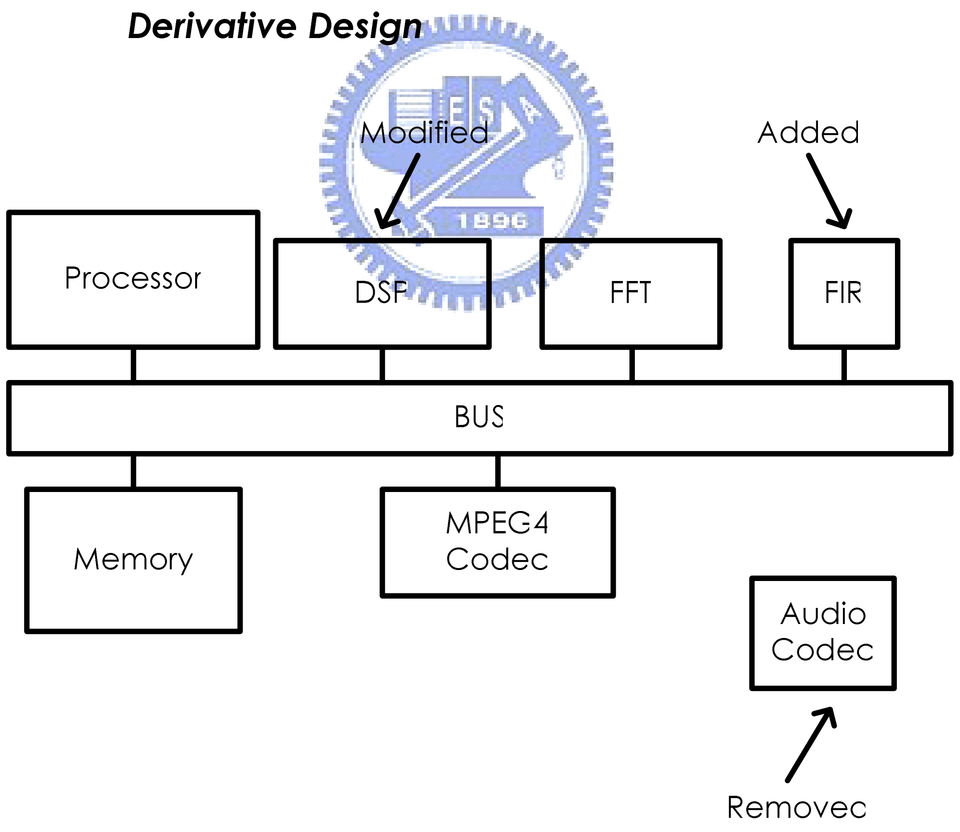


Figure 2.1.1: An Example of ASIC Design

Second, the platform-based architecture design is more flexible than application-specific architecture design. An example of platform-based architecture design is given in Figure 2.1.2. The general platform-based architectures typically include a processor, memory, and communication bus [15]. The intellectual property (IP), such as digital signal processing (DSP), fast Fourier transform (FFT), moving pictures experts group (MPEG) coder and decoder, and audio/video compression and decompression, etc, are all designed with the same protocol of bus and available from the IP libraries to the platform by the application demands. For example, the general platform-based architecture with DSP and audio/video compression and decompression can be used for the video applications, or with DSP and FFT filter can be used for baseband communication. In Figure 2.1.2.(a), the reference design is set for the original specific applications. However, when the specific applications or the required functions are changed, the IP block could be modified, added, and removed to reach the derivatives design which is depicted in Figure 2.1.2.(b). Therefore, the major characteristic of platform-based architecture design is to reduce the design time, since all the devices are based on the same protocol of bus and can be integrated quickly. Unlike application-specific architecture design, the platform-based architecture design could extend to execute more applications by including the extra required IP blocks. In the mean time, the power dissipation will be increased when more and more IP blocks included. The idle unused IP blocks will also waste unnecessary power dissipation. However, in current VLSI circuitry for mobile systems, low power considerations are becoming an important subject since battery life and geometry of mobile systems are limited. Besides, the speed performance will be depended on the slower function units, or mismatch between IP blocks and communication bus. In addition, the scarce memory bandwidth problem between IP blocks and communication bus is not solved totally.



(a)

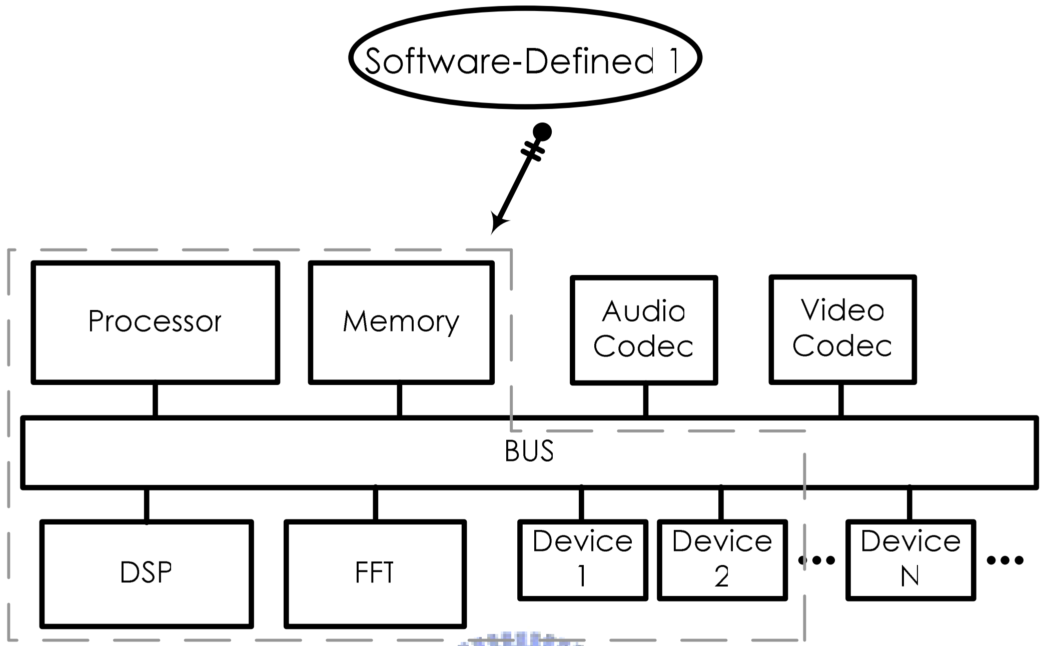


(b)

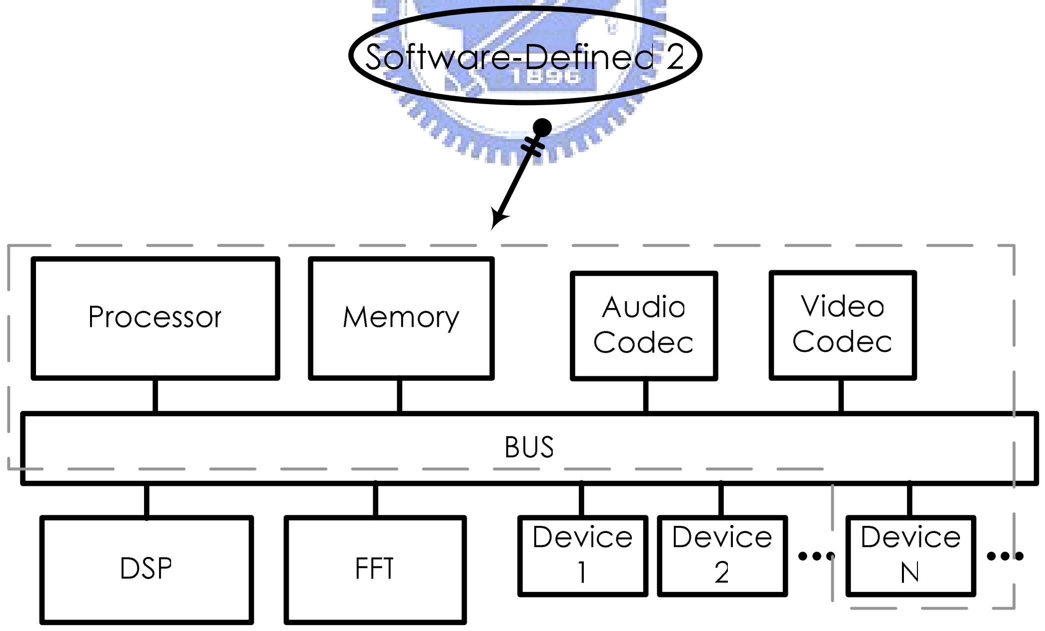
Figure 2.1.2: An Example of Platform-Based Architecture Design

Third, the reconfigurable architecture is similar to the platform-based architecture design. Some general-purpose IP blocks, such as DSP, FFT, audio/video decoder, and so on, are common blocks for many media processing applications. If a platform-based design includes those general IP blocks, the platform-based design can be used to implement a reconfigurable architecture design. An example of reconfigurable architecture design is depicted in Figure 2.1.3. As shown in Figure 2.1.3.(a), the platform is executed an application for DSP where a additional FFT block is needed to accelerate execution rapidly. If the platform is going to execute another application for MPEG decoder where a audio/video coder and decoder is required, the users only have to reconfigure the data path of IP blocks by implementing the software defined radio mechanism, as shown in Figure 2.1.3.(b). One advantage of the reconfigurable architecture is that the used hardware and data path are reconfigurable. This advantage provides a great flexibility for wide different applications. Nevertheless, on the basis of three chief important media characteristics: large available parallelism, little data reuse, and a high computation of memory access ratio, the reconfigurable architecture could not fit in above requirements well since the bandwidth of communication bus is insufficient and then needs the memory hierarchy architecture to solve this bottleneck. Moreover, the inactive unused IP blocks would increase to dissipate needless energy consumption. This also limits to design for the portable systems at the same time.

In conclusion, one of these design methodology styles can be selected to implement that depends on the trade-off of its advantages and disadvantages. As a consequence, the reconfigurable is more suitable to be chosen for the streaming architecture design, because this architecture could provide a significant flexibility in various media processing applications for the software defined radio mechanism.



(a)



(b)

Figure 2.1.3: An Example of Reconfigurable Architecture Design

2.2 Stream Processing Model

Media processing applications are naturally expressed as a sequence of computation kernels that operate on long data streaming [16]. A kernel is a small program that is repeated for each successive streaming element in its input streaming to produce output streaming that is fed to subsequent kernels. Each data streaming is a variable length collection of records, where each record is a logical grouping of media data. In order to illustrate the stream processing model, consider a simple media processing kernel, the finite impulse response (FIR) filter system [17]. An FIR filter is a one dimensional convolution of a small kernel over a long data streaming. Let $y[n]$ represent the output data streaming of an FIR filter. Let T represent the sampled system unit delay which is also equal to the data-rate clock cycle period. At time instant nT the output data sample is given by the following equation describes the operation:

$$y[n] = \sum_{k=0}^{M-1} b_k \bullet x[n-k] \tag{2.1}$$

where M represents the number of taps in the filter, b_k represents the nth tap coefficient, and $x[n]$ represents the input data sample at time instant nT. The direct-form FIR structure of the difference equation of Equation 2.1 could be realized as shown in Figure 2.2.1. A more simplified structure to be expressed to understand the FIR filter system as well as the stream processing model is also shown in Figure 2.2.2. In both Figure 2.2.1 and Figure 2.2.2, solid arrows stand for data streaming, and ovals stand for computation kernels. Therefore, for example, a record could represent an input data streaming in an FIR filter application. A data streaming, then, could be a collection of hundreds of these input data streaming.

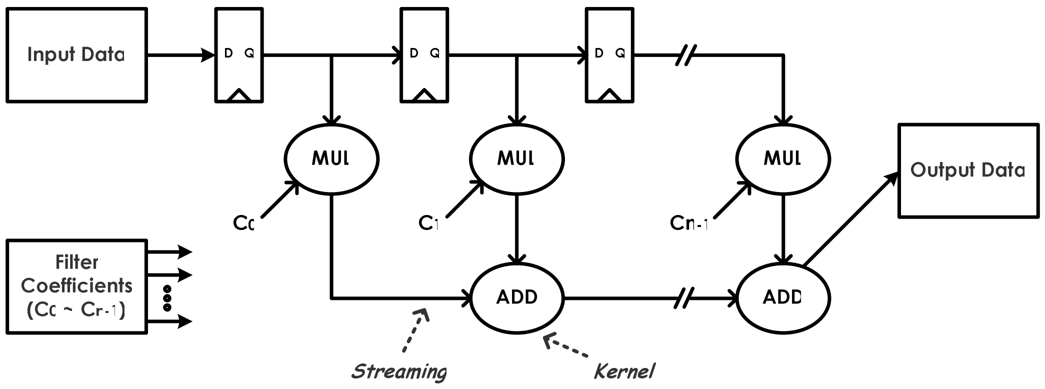


Figure 2.2.1: Direct-Form Realization of an FIR Filter System

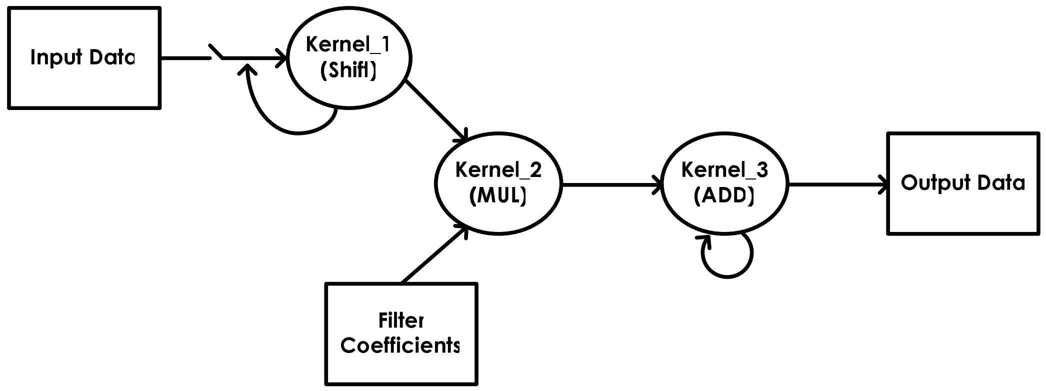


Figure 2.2.2: A Simplified Structure of an FIR Filter System

Records within a data stream are accessed sequentially and processed identically. This greatly simplifies the movement of data through a media processor by allowing the instruction overhead to be amortized over the length of these homogeneous data streaming.

Kernels naturally expose the coarse-grained control parallelism in media processing applications, as they form a pipeline of tasks. Multiple kernels can therefore operate in parallel on different sections of the application's data. The first kernel in the pipeline would produce output streaming that would then be passed to the next kernel. As the next kernel operates on those streaming, the original kernel could operate on the next set of input data. Finally the memory bandwidth demands of media processing application can also be met using this streaming model. Since all data is organized as streaming, single memory transfer operations initiate long transfers with little control overhead that can be optimized for bandwidth.

By organizing media processing applications in this stream processing model, the following characteristics that were enumerated in the previous chapter are exposed: data parallelism is abundant, very little data is reused, and many operations are required per memory reference. Large available parallelism is that operations on one streaming element are largely independent of the others, so they can exploit lots of parallelism and tolerate lots of latency. Little data reuse is that every streaming element is read exactly once from memory and is not revisited, resulting in poor cache performance. A high computation to memory access ratio is that large amounts of arithmetic operations per memory reference are required for each streaming element read from memory. These properties can easily be exploited by a media processor designed to operate on data streaming. The abstraction of a data streaming maps naturally to the streaming data types found in media processing applications. The inputs to most media processing applications are already data streaming and the expected outputs are data streaming as well. Streaming exposes the fine-grained data parallelism inherent in media applications as well. Each record of a streaming will be processed identically, so multiple records can be processed in parallel using the same instructions.

Therefore, any media processing applications could be organized as the stream processing model, such as shown in Figure 2.2.2, our proposed architecture with reconfigurable characteristic could then utilize the software defined radio mechanism to rearrange the architecture system to suit for implementing various media processing applications flexibly.

2.3 Related Research

On the basis of both the streaming architecture and the processing-in-memory have been recommended as an effective architecture for the media processing applications, relative topics of recent research about the steaming architecture are Imagine Stream Processor, Smart Memories, and Processing-In-Memory. In this section, a brief overview of these three architecture solutions will be introduced.

First, Imagine Stream Processor is a programmable single-chip processor that supports the stream programming model. The Imagine architecture supports 48 ALUs organized as 8 single instruction multiple data (SIMD) clusters. Each cluster contains 6 ALUs, several local register files, and executes completely static very long instruction word (VLIW) instructions. The stream register file is the nexus for data transfers on the processor. The memory system, arithmetic clusters, host interface, microcontroller, and network interface all interact by transferring streams to and from the stream register file.

Imagine Stream Processor is a coprocessor that is programmed at two levels: the kernel-level and the stream-level. Kernel functions are coded using KernelC, whose syntax is based on the C language. Kernels may access local variables, read input streams, and write output streams, but may not make arbitrary memory references. Kernels are compiled into microcode programs that sequence the units within the arithmetic clusters to carry out the kernel function on successive stream elements. Kernel programs are loaded into the microcontroller's control store by loading streams from the stream register file. At the application level, Imagine Stream Processor is programmed in StreamC. StreamC provides basic functions for manipulating streams and for passing streams between kernel functions.

Second, Smart Memories is a multiprocessor system with coarse grain reconfiguration capabilities. Processing units in this system are in form of “Tiles” which when put together in groups of four, form “Quads”. Interconnecting these elements is done in a hierarchical manner: a set of Inter-Quad connections provide communication facilities for Tiles inside a Quad, while a mesh interconnection network connects Quads together. Tiles inside a Quad share the network interface to connect to outside world.

Each Tile in the Smart Memories system is consisted of four major parts: Two processor cores, a set of configurable memory mats, a cross bar interconnect, and Load/Store unit. Either or both of the processors inside the Tile can be easily turned off allowing a Tile to be just a memory resource, and saving power, in the case that excess processing power is not required.

Third, Processing-In-Memory architectures that integrate processor logic into memory devices offer a new opportunity for bridging the growing gap between processor and memory speeds, especially for applications with high memory-bandwidth requirements. The data-intensive architecture system combines processing-in-memory memories with one or more external host processors and a processing-in-memory-to-processing-in-memory interconnect. Data-intensive architecture increases memory bandwidth through two mechanisms. First, performing selected computation in memory, reducing the quantity of data transferred across the processor-memory interface. Second, providing communication mechanisms called parcels for moving both data and computation throughout memory, further bypassing the processor-memory bus. Data-intensive architecture uniquely supports acceleration of important irregular applications, including sparse-matrix and pointer-based computations.

In summary, both the streaming architecture and the processing-in-memory solutions commonly occupy an enormous physical area to trade for the data and processing parallelism without incorporating the well-designed power management equipments. On the other hand, low power considerations have been becoming an essential concern in contemporary VLSI design for portable systems. Therefore, a low power controlling mechanism for the next generation media streaming processor architecture will be an emergency issue to investigate to provide an advance in the operating time as well as the power dissipation in the future media processing applications.

2.4 Low Power Considerations

Since the power dissipation has turned out to be a significant design concern in modern VLSI design, high power dissipation would incur expensive package and significant cool cost. On the other hand, the power-aware devices, such as laptops, mobile phones, and handhelds devices, etc, have limited advance in battery technology. It is almost a necessity to make the reduction and control of the power dissipation in high performance digital product designs.

Techniques for the control and reduction of power dissipation can be divided into two main categories: static and dynamic [18]. Static techniques are typically applied during the circuit design phase and do not change during the operation of the circuit. Dynamic techniques allow the dynamic control of certain functional blocks of the design during functional operation. Dynamic techniques are involved setting certain functional blocks of the chip into low leakage mode when they are in idle mode. Furthermore, if the architecture, such as the platform-based architecture, and the reconfigurable architecture, etc, includes the power management unit, system can scale the power and performance of this architecture. When the required performance is high, more function units on the architecture will be active and consume larger power. While the power consumption of the architecture has to be reduced, system can reduce the active hardware and turn them off with power management unit. So the power and performance of this architecture can be scalable if there is power management unit on the architecture.

2.5 Media Streaming Processor Architecture

By reason of the streaming architecture with software defined radio mechanism has been suggested as an effective architecture for the media processing applications, our proposed improved media streaming processor architecture is depicted in Figure 2.5.1. This architecture primarily includes five components: ALU Cluster, stream register file, distributed memory management unit, power management unit, and controller. In the mean time, there is also an architecture simulator that has been built a simulation environment for this architecture.

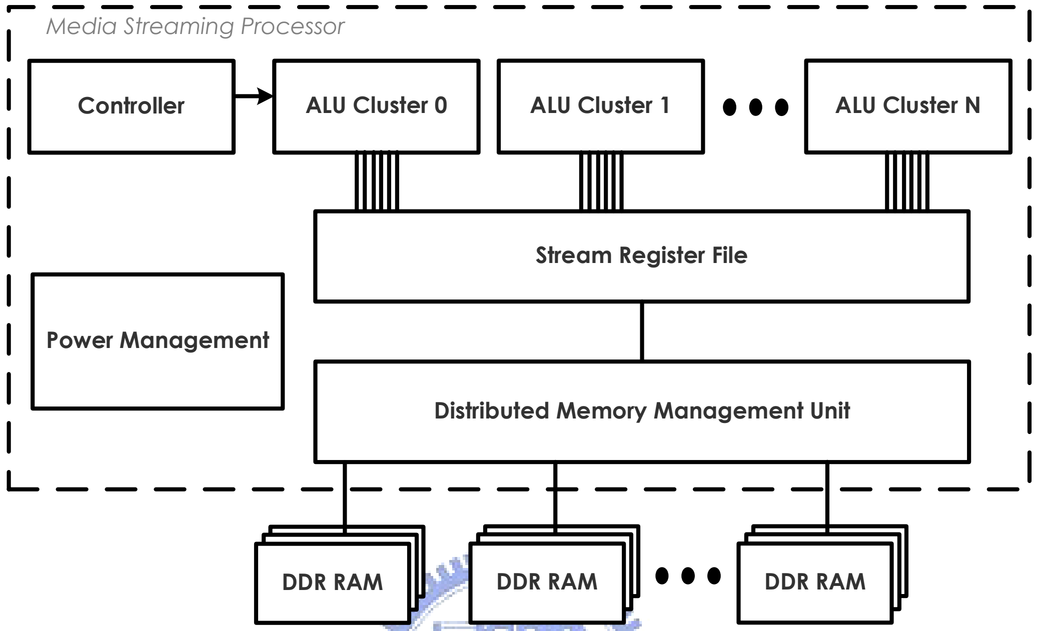


Figure 2.5.1: Media Streaming Processor Architecture

In general, the architecture simulator is used to determine the number and the size of ALU Cluster, stream register file, and off-chip memory to be used efficiently, after simulating a specific media processing application. Therefore, the simulation results from the architecture simulator would then be used with the software defined radio mechanism to reconfigure the media streaming processor. This step would let the system to achieve the best performance that trade between the operating time and the power consumption. The ALU Cluster is used to shorten operating time for the kernel computation. There are also high speed storage units that are embedded in the ALU Cluster to reduce the use of the scarce global communication bandwidth. A controller decodes instructions and then controls the overall operation of all ALU Clusters. The stream register file is a memory organized to handle streaming and could hold any number of streaming of any length. The steam register file supports data streaming transfers between the ALU Cluster and off-chip memory, such as double data rate (DDR) random access memory (RAM), so the recirculation of streaming through the stream register file minimizes the use of scarce off-chip data bandwidth in favor of global register bandwidth. Distributed memory management unit is to solve rare data bandwidth, reduce memory copy, and provide dynamic scheduling.

Along with the power dissipation has become an important issue concern in modern VLSI circuitry for the mobile systems, a power management unit with low power techniques for the control and reduction of power dissipation has been integrated in the next generation media streaming processor architecture. Furthermore, the power management unit could scale the performance and power by trading for turn on or turn off the function blocks that depend on the simulation results from the architecture simulator. In addition, the high speed storage units inside the ALU Cluster, stream register file, and off-chip memory DDR RAM are formed the memory bandwidth hierarchy architecture. This is in contrast to conventional architectures which use less efficient global register bandwidth when local bandwidth would suffice, in turn forcing the use of more off-chip bandwidth.

These main components on the improved media streaming processor and a customized architecture simulator have been under developing at SoC Laboratory by students and faculty. In this thesis, I am chiefly responsible for the design of an ALU Cluster micro-architecture, because ALU Cluster is the nexus computation part of the processors and one key factor of increasing high kernel performance. Besides, the major challenges to complete this work are the architecture decision of each component in the ALU Cluster, met trade-off between timing constraint and area constraint, and complicated to micro-architecture implementation. The detail micro-architecture design and the design flow would be described in the following chapters.

CHAPTER 3 DESIGN OF ALU CLUSTER MICROARCHITECTURE

In this chapter, the details of an ALU Cluster micro-architecture are discussed from the decision of each component in the ALU Cluster to the integration of these components in the ALU Cluster. In addition, the dedicated instruction set format for the ALU Cluster is also explained. Finally, the pipeline steps and the overall system operation with pipeline mechanism on the ALU Cluster are described.

3.1 ALU Cluster Block Diagram

In order to improve the conventional processor architecture that poorly handle with the media processing applications, two solution methods primarily to solve these performance bottlenecks appeared on the conventional processor architecture, such as the required high computation throughput and the processor-memory performance gap, are concurrency and locality, respectively. Therefore, the proposed 32-bit ALU Cluster micro-architecture design is mainly based on the Stanford Imagine Stream Processor [19][20] with the consideration of the implementation feasibility. Concurrency is to provide abundant data-level parallelism which refers to the computation on different data elements occurring in parallel as well as the moderate multiple function units in one ALU Cluster. Locality is temporal and refers to reuse of coefficients or data during the execution of computation kernels, or is also a form of temporal locality that exists between different stages of a computation pipeline or kernels. So the temporary high speed storage unit is embedded inside the ALU Cluster that could form the memory bandwidth hierarchy architectures to reduce the unnecessary use of global memory bandwidth to access the high latency off-chip memory frequently. The block diagram of the ALU Cluster micro-architecture is shown in Figure 3.1.1.

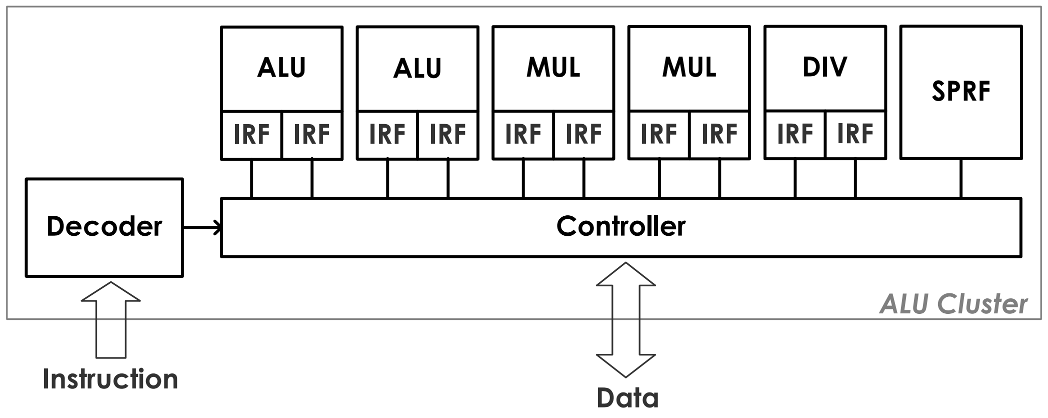


Figure 3.1.1: ALU Cluster Architecture Block Diagram

From Figure 3.1.1, the ALU Cluster architecture are primarily included several arithmetic units, high speed storage units, such as intra-register-file (IRF) and scratch pad register file (SPRF), a decoder, and a controller. The arithmetic units are contained two ALUs, two multipliers, and one divider, supporting to process the large available parallel data simultaneously. Most media processing applications are well suitable for this mixture of arithmetic units. In addition, the back-end simulation results based on the contemporary process technology and the standard cell library could decide the number of parallel arithmetic units. Every arithmetic unit in the ALU cluster is embedded a high speed 32-entry IRF unit for each input. These IRF units mainly are kept to store the temporal intermediate results of computation during executing on streams of data and greatly reduce the usage of the required off-chip memory bandwidth. This allows memory bandwidth to be used efficiently in the sense that expensive and communication limited global memory bandwidth is not wasted on the arithmetic units where inexpensive local memory bandwidth is easy to provide and use. The 64-entry SPRF unit is also an extra high speed storage unit to offer the spills of recirculation of temporary computing data. A decoder fetches instructions and sends the decoded results to the controller. A controller major controls the overall operations of the ALU Cluster. The details of the micro-architecture of these components in the ALU Cluster architecture would be described in the following sections.

3.2 Instruction Set Format

The architecture of instruction set format for the ALU Cluster is similar to a VLIW-like instruction format, as shown in Figure 3.2.1. The instruction set format is major composed of fields for the total arithmetic units (ALU units, MUL units, and DIV unit) used in the ALU Cluster. In addition, each arithmetic unit field is further subdivided into sub-field, and each sub-field is contained by the two input sources and their read addresses, the one output destination and its write address, and the executed operation.

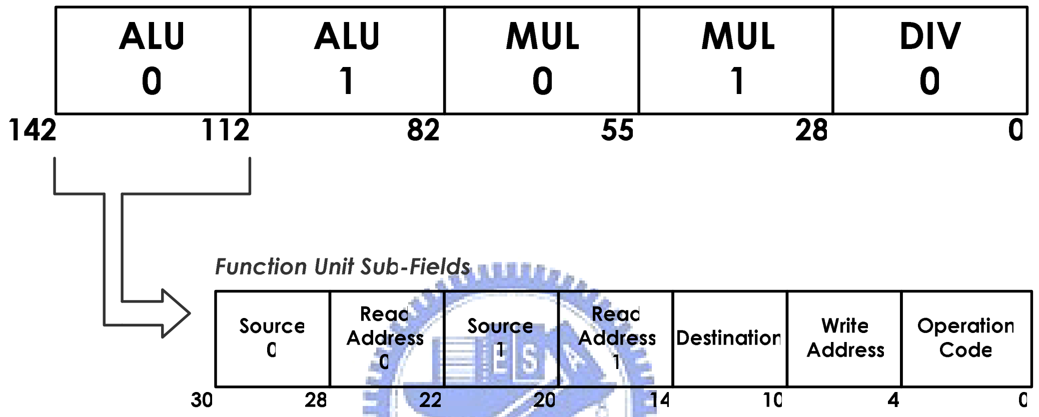


Figure 3.2.1: Instruction Set Format

The input source reads from three: off-chip data memory, self IRF unit, and SPRF unit. In the same way, the output destination writes back to three: off-chip data memory, one of all IRF units, and SPRF unit. The length of address for input source or output destination is determined by the size of maximum storage unit, for example, one of the off-chip data memory, IRF unit, or SPRF unit. The total executable operation types are depended on different type of arithmetic units, so the length of operation code is also depended on the type of arithmetic units. For example, the length of operation code for the ALU unit, the MUL unit, and the DIV unit is 4-bit, 1-bit, and 2-bit, respectively. Details of operation types of this part will be explained in the next section.

The whole length of instruction set would be determined by the total numbers of different type of arithmetic units. The length of the sub-instruction set of each arithmetic unit would be primarily determined by the length of executable operation code. Therefore, for example, the length of the ALU unit, the MUL unit, and the DIV unit is 30-bit, 27-bit, and 28-bit, respectively, and the whole length of instruction set is

142-bit. Details summary of the defined microcode in instruction set for the input source, the output destination, and the executable operation of each arithmetic unit are summarized in Appendix A.

3.3 ALU Cluster Function Units

In this section, the micro-architecture design of each function unit in the ALU Cluster would be discussed in the following subsections. A more detailed view of function unit, which is contained an arithmetic unit and its associated register files, is shown in Figure 3.3.1. Most arithmetic units have two data inputs, and output bus. Data in the function unit is temporary stored in the IRF units. These function units are developed with a number of design goals in mind, including the trade-off between physical area, data throughput, and operation latency. As a consequence, in order to reduce the design complexity and enhance the implementation feasibility, Synopsys DesignWare [21] with UMC 0.18 um 1P6M CMOS process and Artisan SAGE-X standard cell library [22] would be used with the intention of shrinking the time of design process.

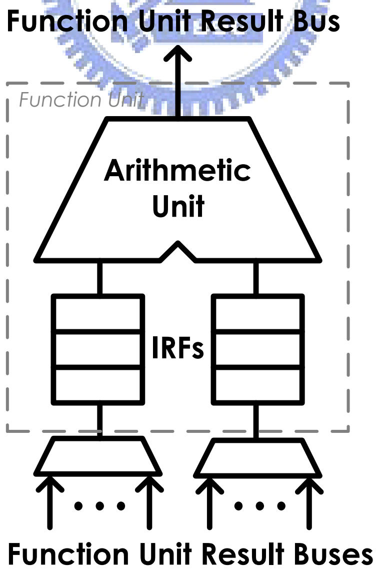


Figure 3.3.1: Function Unit Details

3.3.1 ALU Unit

An ALU Cluster contains two ALU units that could execute the operations, such as the addition, absolute, logical operation, shift, and comparison instructions, listed in Table 3.3.1.1. Many of these operations support for 32-bit signed integer instructions. These operations could be implemented by using Synopsys DesignWare building block IP, for example, DW01_add, DW01_absval, DW01_ash, and DW01_cmp6. The results of synthesis implementation of available IPs are listed in Table 3.3.1.2.

Table 3.3.1.1: The Operations Correspond to the ALU Unit

Operation	*Description*
ADD	Add
SUB	Subtract
ABS	Absolute value
AND	Bitwise AND
OR	Bitwise OR
XOR	Bitwise XOR
NOT	Bitwise invert
SLL	Logical shift left
SRL	Logical shift right
SRA	Arithmetic shift right
LT	Less-than
GT	Greater-than
EQ	Equal

The definition of slack is that the clock period subtracts the library setup time and the data arrival time. Thus, the larger slack value means that more timing margin to complete the execution within one clock cycle. Additionally, the synthesis implementation of available IP only takes the gate delay into consideration without the wire delay, which the wire delay has been grown a chief critical timing issue in the continued scaling of modern VLSI technique [23]. So the larger slack value is better consideration to suitable for the place and route process. The initial clock period is set to 8 ns as well as 125MHz. Besides, the definition of gate count is that the synthesis area of design is divided by the two-input NAND gate area provided by UMC 0.18 um CMOS process with Artisan standard cell library. From Table 3.3.1.2, in order to meet the best optimization between execution time and physical area, therefore, the fast carry-look-ahead architecture is chosen for DW01_add, the carry-look-ahead architecture is chosen for DW01_absval, the 2:1 inverting multiplexers and 2:1 multiplexers architecture is chosen for DW01_ash, and the carry-look-ahead architecture is chosen for DW01_cmp6. Finally, the ALU unit is designed as 2-stage pipeline architecture. The first stage is to fetch the data of two inputs from the controller and then decide which operation to be executed from operation code. The second stage is to complete the assigned execution.

Table 3.3.1.2: Synthesis Results Correspond to Different Architecture

IP	*Implementation*	*Slack*	*Gate Count*
DW01_add	rpl[1]	0.56	244
	cla[2]	2.47	327
	clf[3]	5.63	456
	bk[4]	6.16	459
	csm[5]	6.09	705
	rpcs[6]	2.33	342
DW01_absval	rpl	2.61	229
	cla	3.25	234
	clf	3.14	232
DW01_ash	mx2[7]	6.30	916
	mx2i[8]	6.09	739
	mx4[9]	5.95	975
	mx8[10]	5.16	747
DW01_cmp6	rpl	3.36	235
	bk	5.71	224
	cla	6.34	196

3.3.2 MUL Unit

An ALU Cluster contains two MUL units that could execute the multiplication operation, and the executable operation is listed in Table 3.3.2.1. Like the ALU unit,

1. rpl = ripple-carry
2. cla = carry-look-ahead
3. clf = fast carry-look-ahead
4. bk = Brent-Kung
5. csm = conditional-sum
6. rpcs = ripple-carry-select
7. mx2 = 2:1 multiplexers
8. mx2i = 2:1 inverting multiplexers and 2:1 multiplexers
9. mx4 = 4:1 and 2:1 multiplexers
10. mx8 = 8:1, 4:1, and 2:1 multiplexers

the MUL unit also executes operation to support for 32-bit signed integer instructions. The operation could be implemented by using Synopsys DesignWare building block IP, such as DW02_mult and DW_mult_pipe, too. The results of synthesis implementation of available IPs depended on different pipeline stages are listed in Table 3.3.2.2.

Table 3.3.2.1: The Operation Corresponds to the MUL Unit

Operation	*Description*
MUL	Multiply

From Table 3.3.2.2, the more pipeline stages would make the more slack value, but the gate count from the part of pipeline registers increases more significantly. Therefore, in order to trade between the execution time and the physical area, the 3-stage pipeline with Booth encoding Wallace tree architecture is chosen for DW_mult_pipe contained DW02_mult. Finally, the MUL unit is designed as 4-stage pipeline. The first three stages are to fetch the data of two inputs and complete the multiplication execution. The forth stage is to truncate the outcome of multiplication to maximum or minimum expressible value if overflow or underflow is occurred, respectively.

Table 3.3.2.2: Synthesis Results Correspond to Different Pipeline Stages

IP	*Pipeline Stage*	*Slack*	*Gate Count*
DW_mult_pipe	3	3.95	9873
	4	4.54	12084

3.3.3 DIV Unit

An ALU Cluster contains one DIV unit that could execute the operations, such as the division and square root instructions, and these executable operations are listed in Table 3.3.3.1. Like both the ALU unit and the MUL unit, these operations also support for 32-bit signed integer instructions. On the other hand, the DIV unit is not the key kernel performance concerned, therefore, the DIV unit would be suggested not to be pipelined and by increasing the latencies of the execution to trade for shrinking area. These operations could be implemented by using DW_div and DW_sqrt of Synopsys DesignWare building block IP. The results of synthesis implementation of available IPs are listed in Table 3.3.3.2.

Table 3.3.3.1: The Operations Correspond to the DIV Unit

Operation	*Description*
DIV	Quotient
REM	Remainder
SQR	Square root

From Table 3.3.3.2, in order to minimize the physical area of the DIV unit by means of increasing execution latencies, therefore, the ripple-carry architecture is chosen both for DW_div and DW_sqrt. Finally, the DIV unit is designed as no pipeline architecture but with a latency of 16 clock cycles for the execution of each time.

Table 3.3.3.2: Synthesis Results Correspond to Different Architecture

IP	*Implementation*	*Data Arrival Time*	*Gate Count*
DW_div	rpl	194.34	6628
	cla	83.63	9127
DW_sqrt	rpl	68.23	1585
	cla	44.44	1914

3.3.4 IRF Unit

While the ALU unit, the MUL unit, and DIV unit are supported all of the arithmetic operations in the ALU Cluster, an important non-arithmetic operation is supported by the IRF unit. The IRF unit is a one read port and one write port high speed register file, and the flip-flops are used as the basic storage element for the IRF units, as shown in Figure 3.3.4.1. The storage capacity of IRF unit is 32 words. The multi-level multiplexer trees are taken the place of the single-level multiplexer trees to speed up the combinational circuit's part of multiplexer. The IRF unit could be written one data and read another data within the same clock cycle, and the flip-flops before the read selects of IRF unit enable the register file holding the input values within the IRF units so that data written on one clock cycle could be read correctly by the arithmetic unit in the subsequent clock cycle.

The key function of IRF unit is major kept to store the temporal intermediate results of calculation during executing on streams of data and significantly decrease the usage of the necessary off-chip memory bandwidth. This allows memory bandwidth to be used efficiently in the sense that the high-cost and communication limited global memory bandwidth is not wasted on the function units where the low-priced local memory bandwidth is simple to utilize and offer. In conclusion, all IRF units in the ALU Cluster have a total of 320 words, and provide 8 GB/s of peak memory bandwidth for one ALU Cluster.

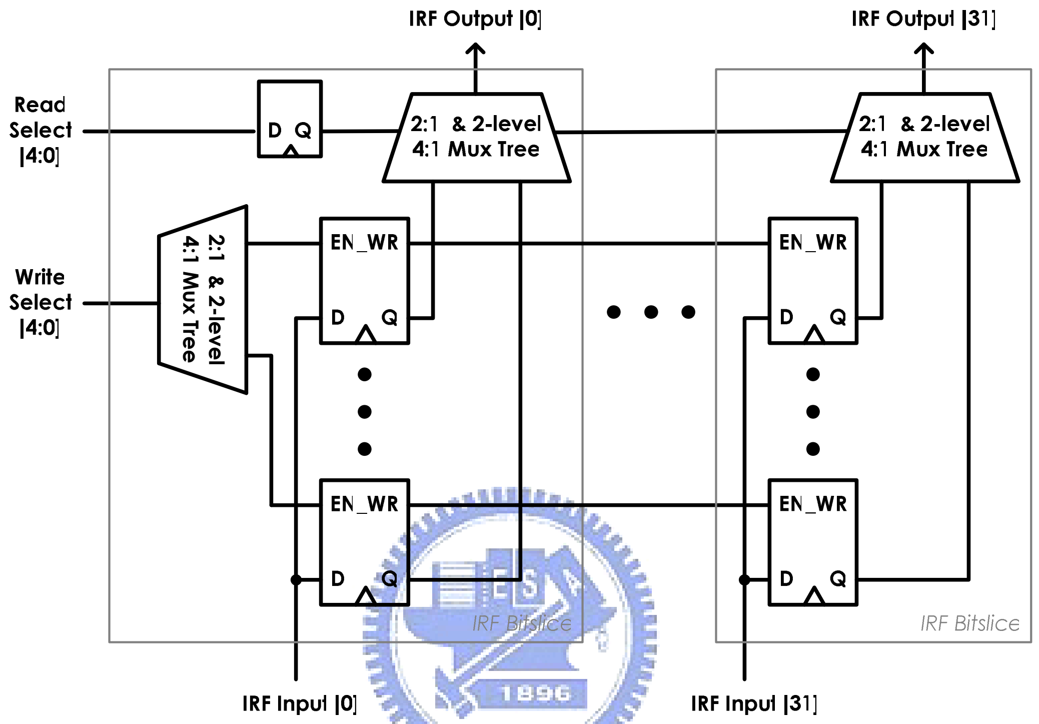


Figure 3.3.4.1: IRF Architecture

3.3.5 SPRF Unit

Another non-arithmetic operation is supported by the SPRF unit. The architecture of SPRF unit is analogous to the architecture of IRF unit except for the size of storage capacity. The SPRF unit is a one read port and one write port high speed register file, and the flip-flops are used as the basic storage element for the SPRF units. The storage capacity of SPRF unit is 64 words, and the SPRF unit could provide 0.8 GB/s of peak memory bandwidth for one ALU Cluster. The SPRF unit could be written one data and read another data within the same clock cycle, and the data written on one clock cycle could be read correctly by the arithmetic unit in the subsequent clock cycle. The primary functions of SPRF unit are to hold some spills from IRF units and store common coefficient parameters.

3.3.6 Decoder Unit

The decoder unit provides to fetch the VLIW-like 142-bit instructions from the off-chip instruction memory, and then decodes these instructions for the controller. First, the fetched instruction is divided into several segments depended on the number of arithmetic units. Second, none operation instruction segments are discarded and then the leftover instruction segments are transformed to the requested binary code type for the controller. Finally, the decoded results from the decoder are sequenced to the controller.

3.3.7 Controller Unit

The controller provides temporary storage to hold the decoded instructions, and then sequences and issues these decoded instructions to the function units during execution. The controller is divided into two parts: the read control and the write control. The part of read control receives the decoded instructions from the decoder, and then acknowledges the storage unit, such as off-chip memory, IRF unit, or SPRF unit, to read out the desired data to the assigned function unit. On the other hand, the part of write control would hold the decoded instructions till the function unit that has finished the execution, and then acknowledges the destined storage unit to be written back the result of computation. The precise timing mechanism and the exact computation data flow are two essentially tasks for the controller to manage the overall operation of the ALU Cluster.

3.4 System Operation

In order to increase computation throughput and decrease operation period, the system operation with pipeline mechanism has been recommended as one of solution ways to achieve these goals. Therefore, as is naturally done in most high performance processors, the ALU Cluster also operates in a pipelined manner to reach higher instruction throughput. The pipeline execution diagram in the ALU Cluster is depicted in Figure 3.4.1. The complete process of pipeline operation to execute one instruction includes from FETCH, DECOED, READ REGISTER, and EXECUTE 1 ~ N, to WRITE BACK.

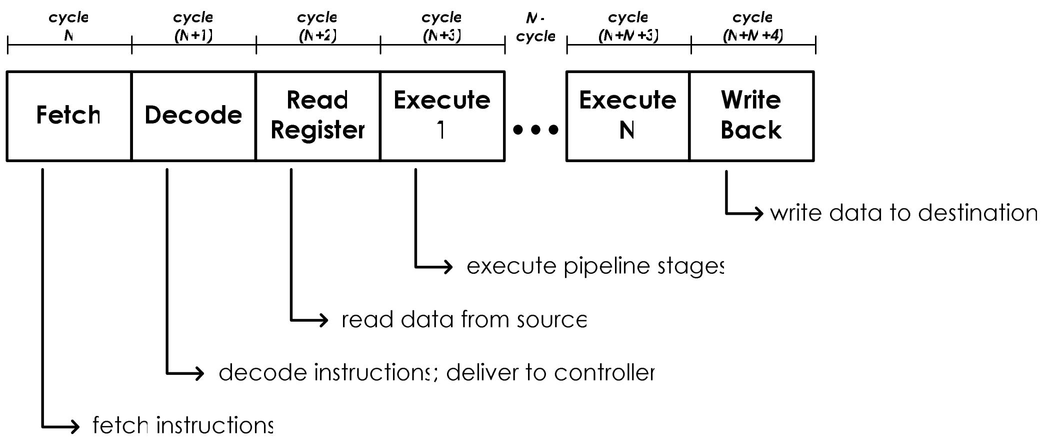


Figure 3.4.1: Pipeline Execution Diagram Details

During the first pipeline stage in the cycle N (FETCH), the decoder fetches and sequences the VLIW-like instructions from the instruction microcode storage. During the decoding stage (DECODE), the decoder decodes the incoming instructions and then delivers the decoded results to the controller. During the register file read stage (READ REGISTER), the controller would manage the data storage unit to be read out the desired data. The desired data major comes from one of the off-chip data memory, self IRF unit, or SPRF unit, and then sends to the dedicated function unit. During the execution stage (EXECUTE), each function unit begins to execute the computing operation if it has been assigned. The duration of executing clock cycle is depended on the types of function unit, for example, the ALU unit is 2 clock cycles, the MUL unit is 4 clock cycles, and the DIV unit is 16 clock cycles. Finally, during the register file write stage (WRITE BACK), the computing results from the function unit would be written back to the assigned data storage unit managed by the controller. Similarly, the assigned data storage unit also mainly comes from the off-chip data memory, one of all IRF units, or SPRF unit.

In summary, while there are perfectly no any hazards happened among the VLIW-like instructions, the sequence pipeline operation mechanism of ALU Cluster is shown in Figure 3.4.2. Although the VLIW-like instructions are scheduled statically and sequenced to the ALU Cluster, any hazards during execution could cause the succession pipeline operation to stall. Thus, if the hazard is encountered, the instructions issued earlier would continue to be executed, but the instructions issued later should be stalled and then be re-executed after the stall condition is no longer valid.

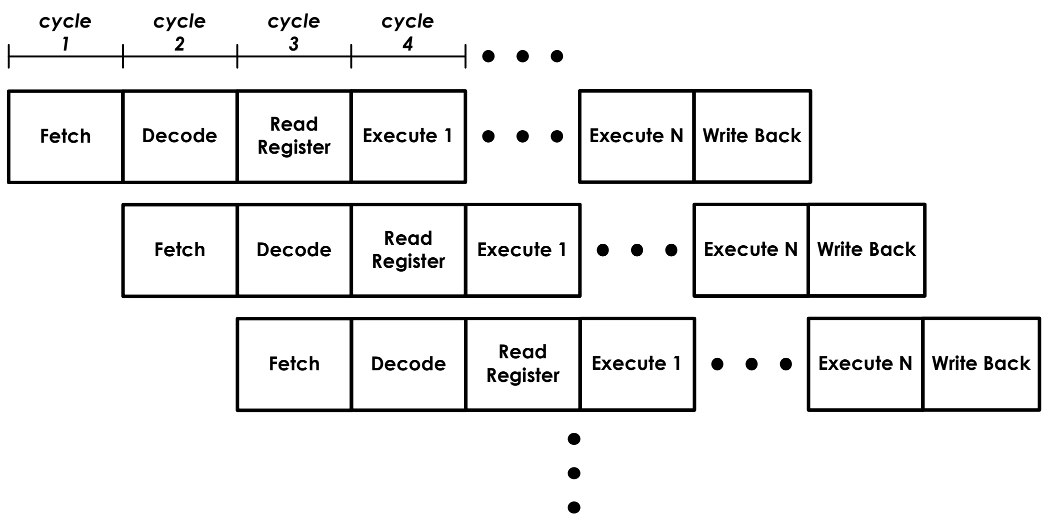


Figure 3.4.2: Sequence Pipeline Operation Diagram

CHAPTER 4 IMPLEMENTATION

In this chapter, the design of an ALU Cluster micro-architecture, described in previous chapter, would be implemented with the cell-based design method. The EDA flow for the implementation of this design is introduced, and the circuit implementation results are listed. The verification results of this work are discussed from the benchmark choice, the chip configuration for simulation, functionality test verification, to the performance evaluation. Finally, the performance comparisons to current related architecture design, implementation of power saving techniques, and a brief summary of this work are also discussed.

4.1 Design Flow

In order to accomplish the implementation of proposed ALU Cluster micro-architecture, from the defined specifications to the die chip achievement, the feasible methods should be provided to complete this work. For most traditional digital circuit design, the computer-aided design (CAD) tools could be supported to deal with these designs. With the help of CAD tools, the time of circuit design process could be shrunk greatly. Besides, the verification and the debug are easily to be detected and handled. A complete digital circuit design flow with the provided standard cell library, for example, the cell-based design flow, is shown in Figure 4.1.1. Three main CAD tools are used to design this work: simulator, synthesizer, and automatic placement and route (APR). In addition, the major steps of design flow include from the architecture design, register transfer level (RTL), gate-level, physical-level, verification, to tape out. The details of these steps are explained in the following:

1. *Architecture design*: This is the initial step to design an integrated circuit (IC). The detail specifications and components of an ALU Cluster should be determined definitely and feasibly.

2. *RTL*: The determined architecture is stylized by using the hardware description language (HDL) code, such as Cadence NC-Verilog [24], to describe the behavior function of each module. The verification of this step is used Novas Debussy [25] to certify the functionality simulation without taking any timing delays into account. Once functionality simulations of RTL do not match to the required specification, the HDL codes should be corrected, or the architecture would be modified until meeting the demands.

3. *Gate-level*: After the verification of functionality simulation is met with the specification of architecture design, the synthesizable RTL codes are synthesized by utilizing the CAD tool, such as Synopsys Design Compiler [21], to the logic cells. The targeted technology process and the essential synthesis constraints would be selected and set to meet the performance requirements. The functionality simulation with considering the gate delays would be performed for the pre-layout verification.

4. *Physical-level*: The synthesizable codes with logic cells would be transformed from the gate level model into the transistor level model in this step. The APR, such as Synopsys Astro [21], could be completed the physical implementation. The basic design flow of APR is included the following sub-steps: the global net connection specification, floor planning setup, timing setup, placement and optimization, clock tree synthesis, global nets connection, routing and optimization, and stream out. The gate delay and wire delay would be taken into consideration when performing the post-layout functionality simulation checks.

5. *Verification*: Another two post-layout verifications are also necessary. One is the design rule check (DRC), and the other is the layout versus schematic (LVS). DRC checks the data of physical layout against the design rules of fabrication, because the design rule document is golden for each design to have to be followed. LVS checks the connectivity of physical layout to its relative schematic circuit netlist. The Mentor Calibre [26] could be used for these verifications.

6. *Tape-out*: The ultimate physical layout would be produced after having gone through the overall design flow, and then it could be fabricated in the foundry.

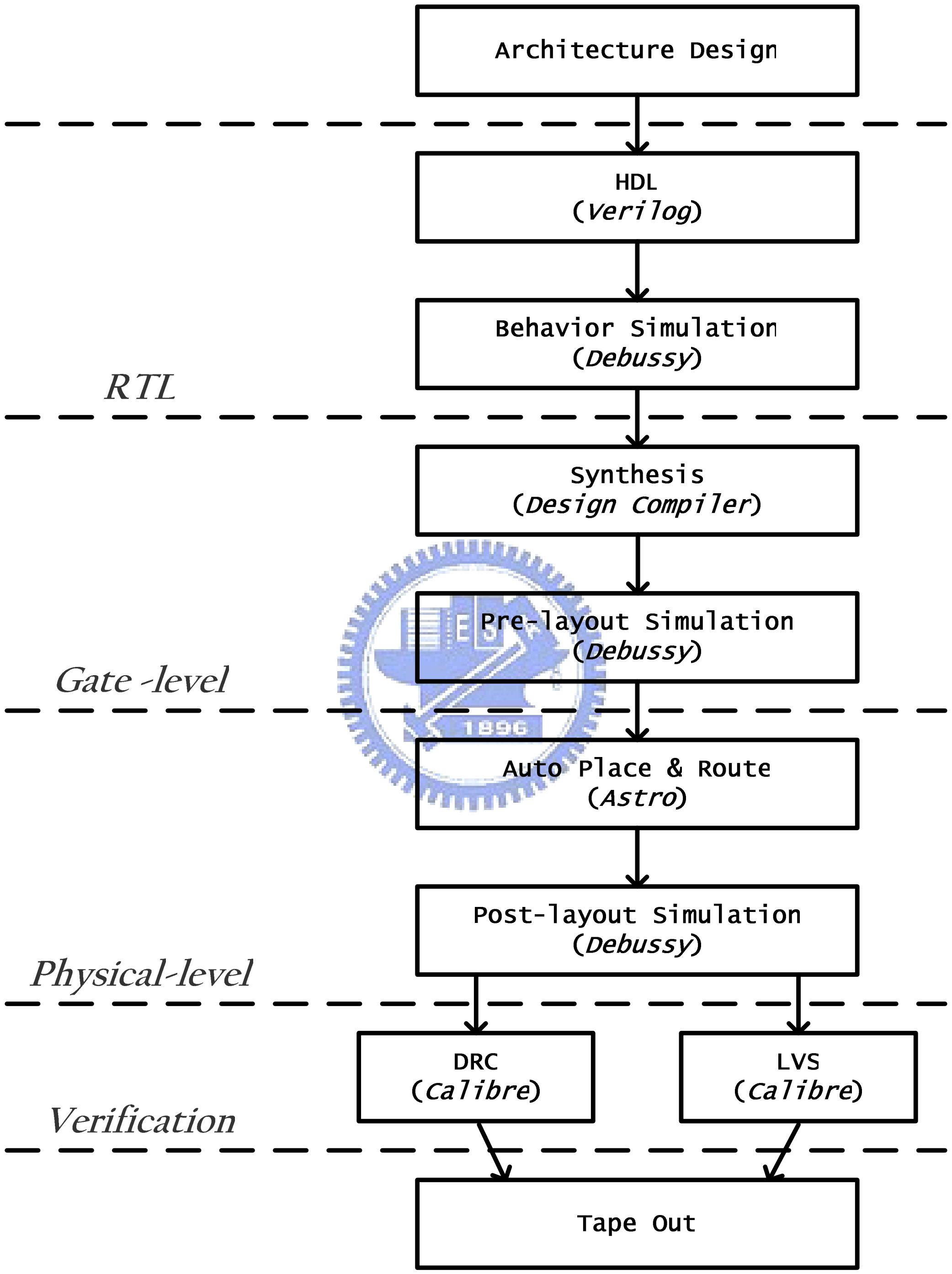


Figure 4.1.1: Cell-Based Design Flow

4.2 Circuit Implementation and Results

The summary of circuit characteristics of this work are listed in Table 4.2.1. UMC 0.18 um CMOS process and Artisan design kit are utilized for the implementation. The post-layout operation frequency of an ALU Cluster is 100MHz. The chip size, core size, gate count, and power dissipation are about 3 mm^2, 2.2 mm^2, 411491, and 968 mW, respectively. There are total fifteen memories included in this work. The four 32 x 128 single port static RAM (SRAM) and one 14 x 128 single port SRAM are the instruction memory, which is stored the instructions for executing operation. The ten 32 x 32 single port SRAM are the data memory, which is stored the required data for program execution. Without these memories contained in this work, the core size, gate count, and power dissipation are near 1.47 mm^2, 255669, and 312 mW, respectively. The physical layout of ALU Cluster is depicted in Figure 4.2.1. The core utilization is close to 88.8%. The floorplan and pad assignment are shown in Figure 4.2.2. There are total 127 input/output (I/O) pads, where 47 input pads, 32 output pads, and 48 power pads. The definition of the I/O ports is summarized in Table 4.2.2. Besides, the die microphotograph of tape-out chip is shown in Figure 4.2.3. The selected package for this chip is CQFP128, and photograph of prototype with package is shown in Figure 4.2.4.

Table 4.2.1: Circuit Summaries

Technology	UMC 0.18um Mixed Signal (1P6M) CMOS Process
Library	Artisan SAGE-X Standard Cell Library
Clock Rate	100 MHz
Chip Size	2.98 x 2.98 mm^2
Core Size (without memory)	2.2 x 2.2 mm^2 (1.8 x 1.2 mm^2)
Gate Count (without memory)	411491 (255669)
Power Dissipation (without memory)	968.35 mW (312.38 mW)
On-Chip Memory	10 32 x 32 single port SRAM 4 32 x 128 single port SRAM 1 14 x 128 single port SRAM
Pad	Input: 47 pins Output: 32 pins Power: 48 pins

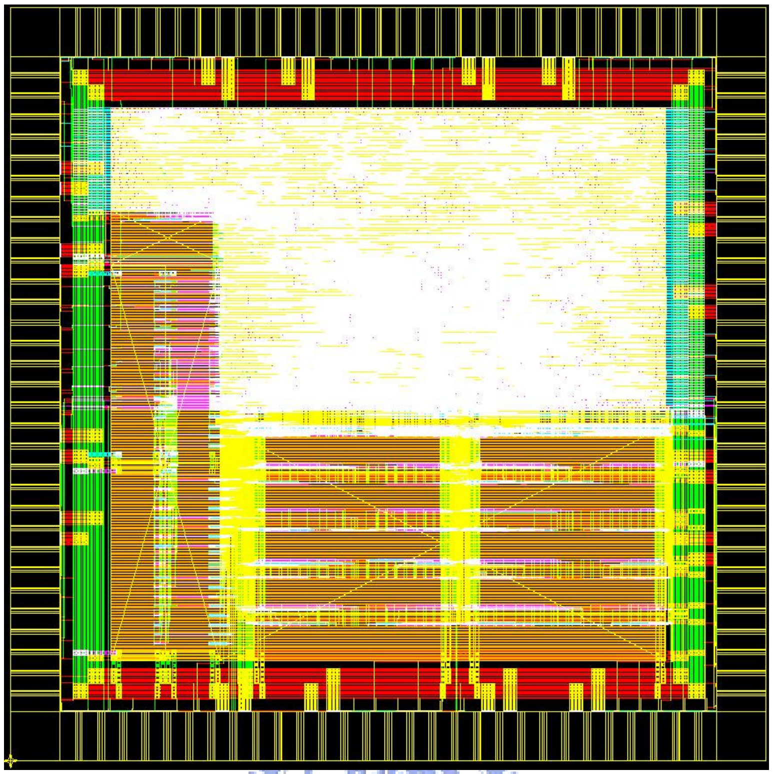


Figure 4.2.1: Layout of the ALU Cluster

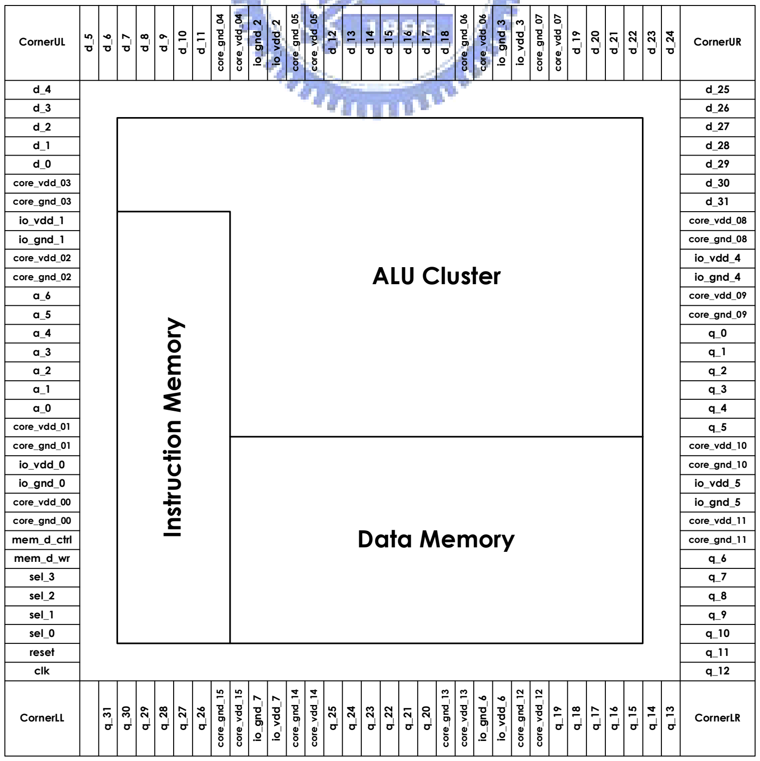


Figure 4.2.2: Floorplan and Pad Assignment

Table 4.2.2: The Definition of the I/O Ports

I/O Port Name	*I/O*	*Signal Description*
clk	Input	The clock signal provides for this chip.
reset	Input	The reset signal provides for this chip.
sel	Input	This is 4-bit width input. To select one of the instruction memories and the data memories to be written, or one of the data memories to be read.
mem_d_wr	Input	This input port decides to write or read the data memory. "1" means that data is written from the off-chip ports to the data memory. "0" means that data is read from the data memory to the off-chip ports.
mem_d_ctrl	Input	This input port decides which source signal controls the data memory to be activated. "1" means that the off-chip port controls the enable signal of data memory. "0" means that the on-chip signal controls the enable signal of data memory.
a	Input	This is 7-bit width input. User can specify the address of instruction memory and data memory by this input port.
d	Input	This is 32-bit width input. User can insert instructions to the instruction memory and data to the data memory by this input port.
q	Output	This is 32-bit width output. User can fetch execution results from the data memory by this output port.
core_vdd & core_gnd	Power	The power supply provides for the core part of chip. There are total 16 pairs of power supply.
io_vdd & io_gnd	Power	The power supply provides for the I/O part of chip. There are total 8 pairs of power supply.

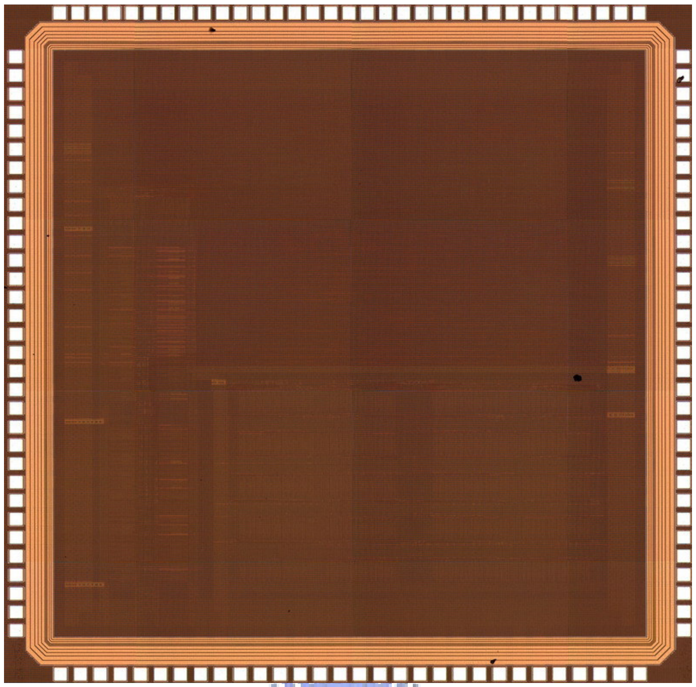


Figure 4.2.3: Die Microphotograph

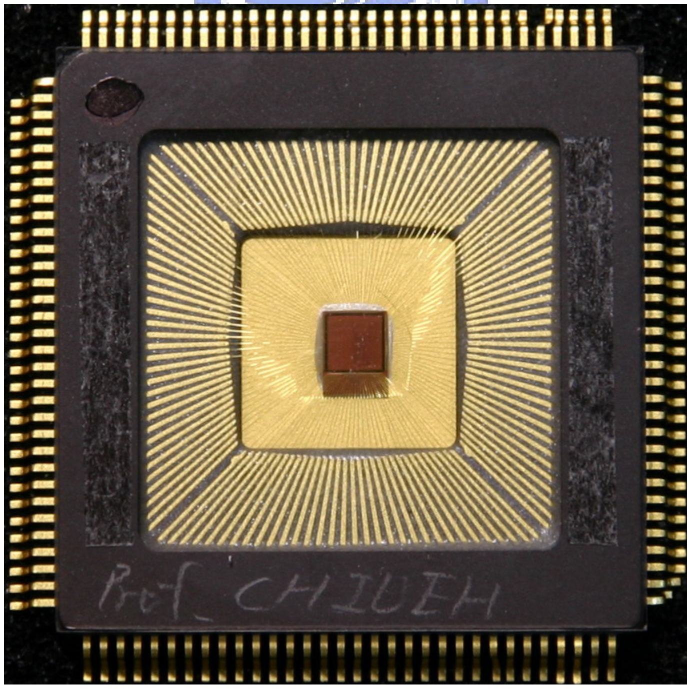


Figure 4.2.4: Photograph of Prototype with Package

4.3 Circuit Verification and Performance Evaluation

In this section, a selected benchmark is used to show the functionality verification of this work. In order to test with feasibility and ease, three steps of test configuration for this chip would be explained. The functionality simulation and the verification during each step of chip configuration would be also described. Finally, the performance evaluation of this work would be discussed.

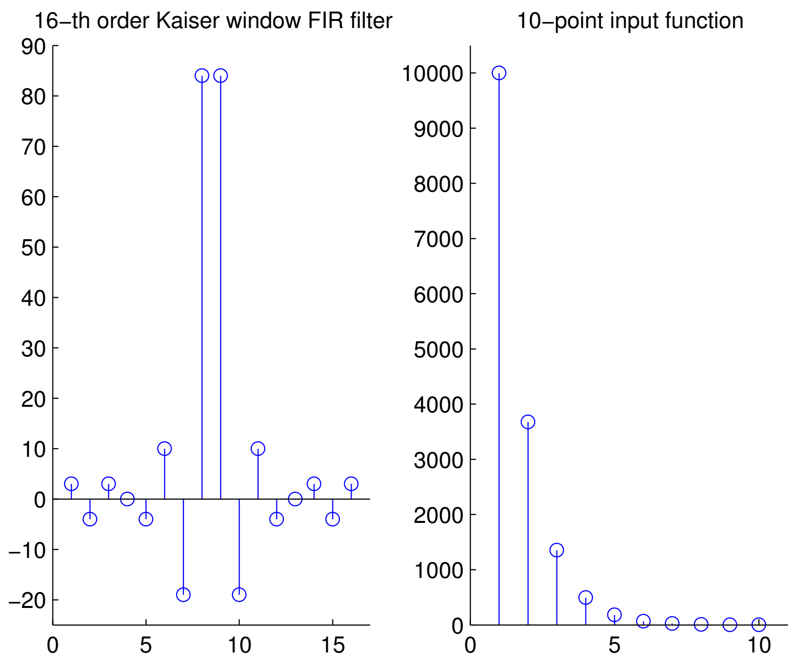
4.3.1 Test Bench: FIR Filter

Owing to media processing applications are easily expressed as a series of computation kernels that operate on large data streaming. As long as any media processing application could be organized as the stream processing model that would be suitable for the ALU Cluster to execute, for instance, the FIR filter system has been introduced in previous chapter and is depicted in Figure 2.2.2. The FIR filter system is chosen as the test bench for the ALU Cluster since it is suitable for one dimensional architecture, needs repeat and high percentage of addition and multiplication, and applies for wide DSP applications, such as matched filtering, pulse shaping, equalization, etc. A brief review of FIR filter system is illustrated in the following. The input-output relationship of linear time invariant (LTI) FIR filter can be described as

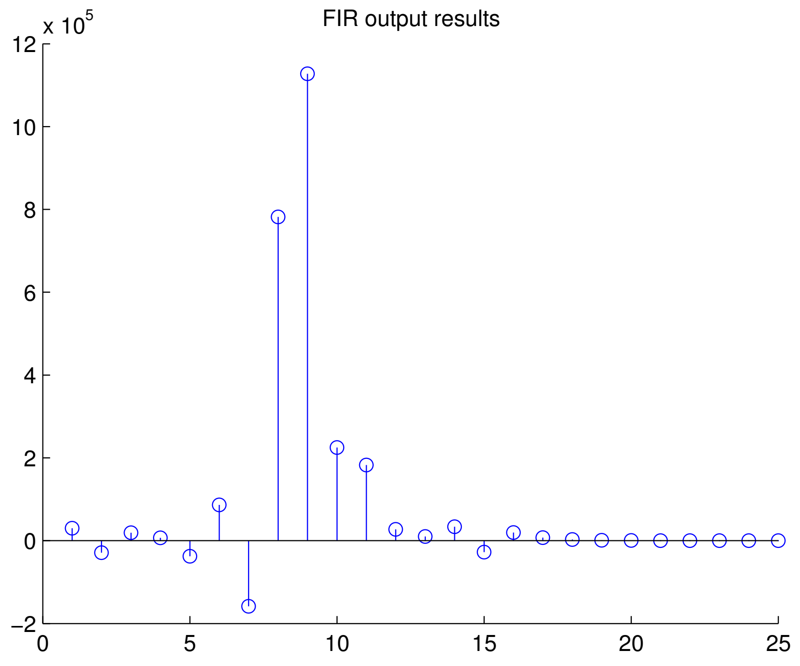
$$y[n]=\sum_{k=0}^{M-1} b_k \bullet x[n-k] \tag{4.1}$$

where M represents the length of FIR filter, b_k's are the filter coefficients, and $x[n\text{-}k]$ denotes the data sample at time instance $[n\text{-}k]$.

Before executing simulation, the dimension of input and filter coefficients should be determined. As shown in Figure 4.3.1.1(a), the filter coefficients are the sixteen-tap Kaiser window FIR bandpass filters, and the input is an exponential function with ten sampling points. MathWorks Matlab [27] is used to simulate the FIR filter system described above in advance, and the results of simulation are shown in Figure 4.3.1.1(b). This step is in order to make sure the results of FIR filter execution under calculating in the ALU Cluster that could be compared to the results of Matlab simulation to verify whether the functionality operations of this chip work correctly or not.



(a)



(b)

Figure 4.3.1.1: Filter Coefficients, Input Data, and Executed Results of the FIR Filter

4.3.2 Functionality Verification

One of design goals for the ALU Cluster is to process the abundant parallel data, so the total numbers of input pads and output pads are enormous significantly. However, the SRAM, such as the instruction memory and the data memory, is utilized to replace and reduce the most of input pads and output pads. Therefore, for the testability and feasibility of this chip, the ALU Cluster would be operated in three different modes: WRITE Mode, EXECUTION Mode, and READ Mode. When this chip is ready to execute programs, it would be operated in the order from WRITE Mode, EXECUTION Mode, to READ Mode. The detail actions of three modes would be described in the following:

1. *WRITE Mode*: The first step is to insert the instructions and the required data into the instruction memory and the data memory, respectively, from the input port "d." With combination of the other input ports, such as "sel," "a," "mem_d_wr," and "mem_d_ctrl," to be controlled and set, user could determine one of the instruction memory or the data memory is the writing target. The value of control signals for memory in this mode are:

   mem_d_wr = high & mem_d_ctrl = high

2. *EXECUTION Mode*: After inserting the instructions and the required data into the dedicated memory, the second step is that the ALU Cluster could be begun to execution the assigned programs. In this mode, the input ports, such as "sel" and "a," are used to control the instruction memory to issue the instructions, and the other input ports, such as "mem_d_wr" and "mem_d_ctrl," are used to set the data memory to be controlled by the on-chip signals. The value of control signals for memory in this mode are:

   mem_d_wr = low & mem_d_ctrl = low

3. *READ Mode*: In the third step, user could read out the data from the data memory for testing after the assigned program execution has been finished. With combination of the input ports, such as "sel," "a," "mem_d_wr," and "mem_d_ctrl," the computed data could be read out from the data memory to the output port "q." The logic analyzer could be utilized to confirm that whether the computed results are accurate or not. The value of control signals for memory in this mode are:

   mem_d_wr = low & mem_d_ctrl = high

In order to verify functionality of this work, there are three modes to complete a program execution has been described in previous paragraph, and the steps of functionality verification would also follow in this order to be discussed. All functionality verifications are under the environment of post-layout simulation, and the maximal operation frequency is 90.9 MHz for executing the FIR filter system. The overall operation modes are shown in Figure 4.3.2.1.

Before executing the assigned programs, the WRITE Mode is executed firstly while having set the input ports, such as "mem_d_wr" is high and "mem_d_ctrl" is high. The ALU Cluster during the WRITE Mode is shown in Figure 4.3.2.2. In this mode, not only instructions are inserted into the instruction memory, but also the filter coefficients and the input data of the FIR filter are inserted into the data memory. Figure 4.3.2.3 and Figure 4.3.2.4 are shown the insertion of filter coefficients and input data, respectively. In addition, the assembly code of overall instructions for the execution of this test bench is summarized in Appendix B.

After having completed the WRITE Mode, the EXECUTION Mode could be started. Figure 4.3.2.5 is shown the ALU Cluster operated in the EXECUTION Mode after having set the input ports, such as "mem_d_wr" is low and "mem_d_ctrl" is low. The pre-stored instructions are fetched from the instruction memory to the decoder, and then the controller governs overall ALU Cluster to execute the programs. In the mean time, the required input data is read from the data memory, and only the calculated results are also written back to the data memory.

Finally, the last step is to verify the results of program execution. After the WRITE Mode has been finished, this chip is entered into the READ Mode with setting the input ports, such as "mem_d_wr" is low and "mem_d_ctrl" is high. Figure 4.3.2.6 is shown the ALU Cluster worked in the READ Mode. To compare the results read from Figure 4.3.2.6 and the results shown in Figure 4.3.1.1(b), there is no difference between these two results. Therefore, the functionality of ALU Cluster is worked correctly.

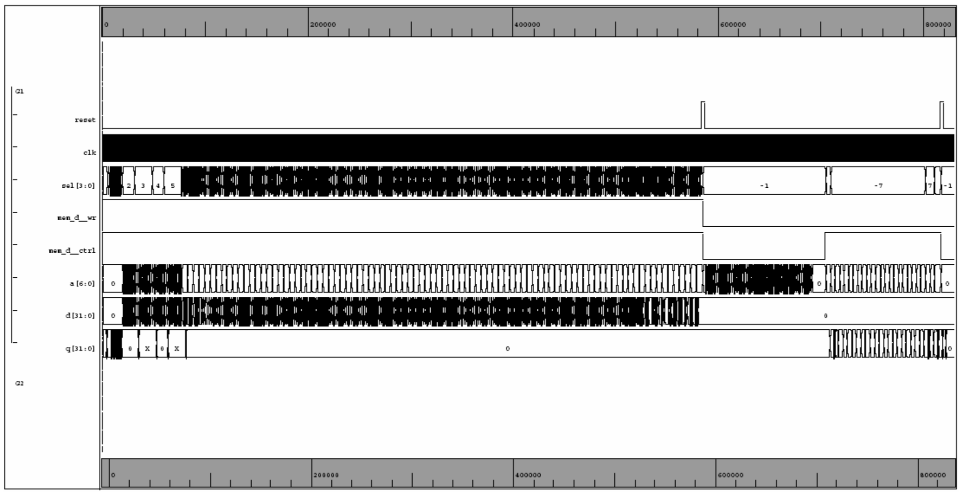


Figure 4.3.2.1: The Overall Operation Flow

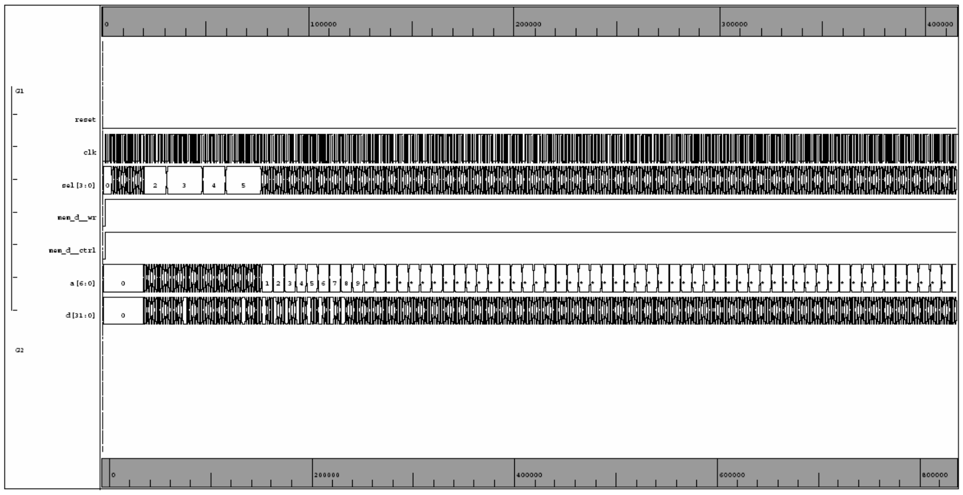


Figure 4.3.2.2: The Operation of WRITE Mode

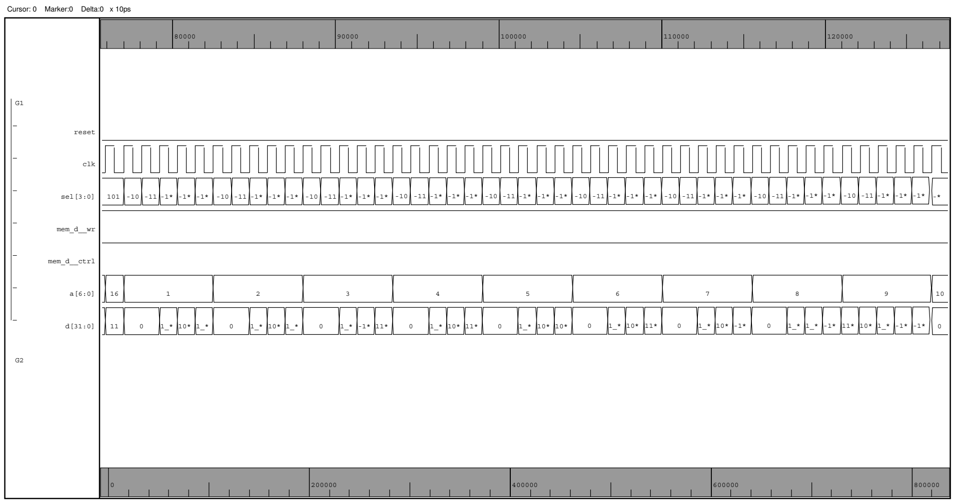


Figure 4.3.2.3: Insertion of Filter Coefficients

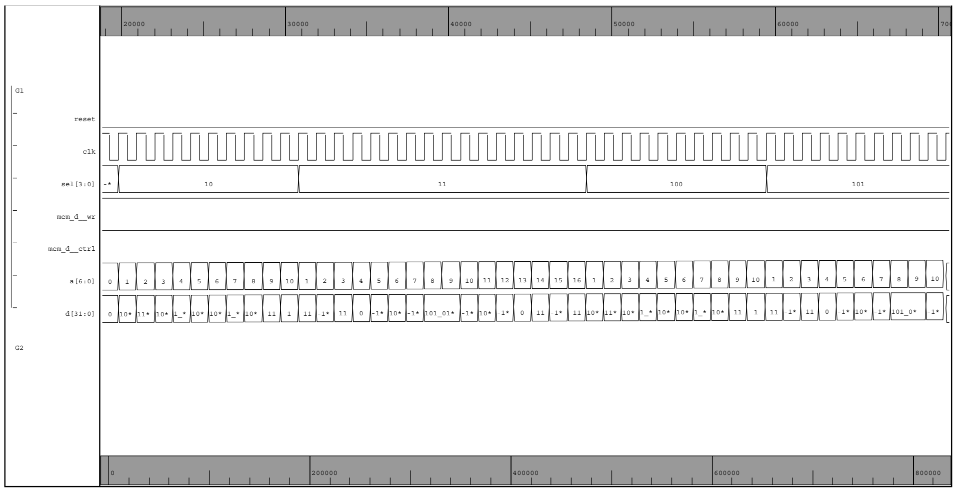


Figure 4.3.2.4: Insertion of Input Data

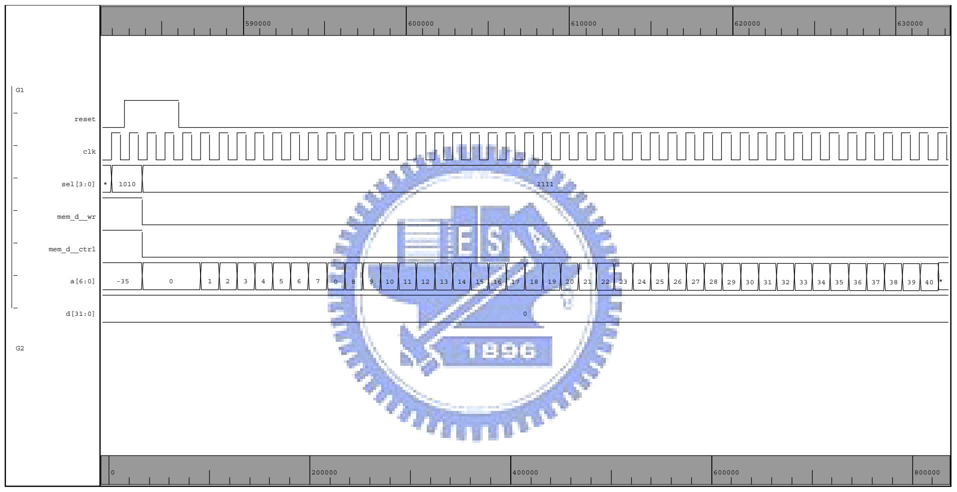


Figure 4.3.2.5: The Operation of EXECUTION Mode

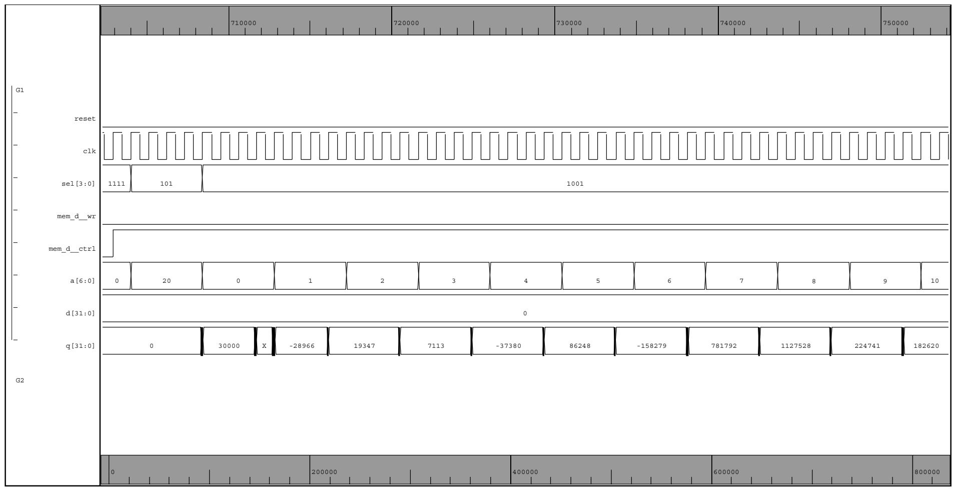


Figure 4.3.2.6: The Operation of READ Mode

4.3.3 Performance Evaluation Results

After having completed the execution of FIR filter system in the ALU Cluster, the results of performance evaluation about the code utilization and the memory utilization could be acquired. The detail performance evaluations are discussed in the following.

Figure 4.3.3.1 is shown the code utilization of each arithmetic unit. It takes total 93 instructions for the ALU Cluster to finish the FIR filter simulation. For each arithmetic unit, it takes 60, 75, 80, 80, and 0 instructions for the ALU_0 unit, ALU_1 unit, MUL_0 unit, MUL_1 unit, and DIV_0 unit, respectively, to complete the program execution. Additionally, the code utilization of the ALU_0 unit, ALU_1 unit, MUL_0 unit, MUL_1 unit, and DIV_0 unit is 64.5%, 80.6%, 86%, 86%, and 0%, respectively. Therefore, the code utilization of ALU Cluster is about 63.4%. Besides, it takes 99 clock cycles to complete this simulation, so the clock cycles per executed result output are 3.96.

Figure 4.3.3.2 is shown the memory utilization about the capacity usage in the ALU Cluster. The entry size of IRF unit and SPRF unit is 32 and 64, respectively. It needs 10 and 12 reused entries for each IRF unit in the ALU_0 unit and ALU_1 unit, respectively, 16 and 10 reused entries for each IRF unit respectively in both the MUL_0 unit and MUL_1 unit, 3 reused entries for the SPRF unit, and 0 used entries for each IRF unit in the DIV_0 unit during executing the FIR filter simulation. These results have revealed that the initial decisions of storage capacity of IRF unit and SPRF unit are well sufficient to be provided and used during the execution of FIR filter system.

Figure 4.3.3.3 is shown the memory utilization about the data reference times in the ALU Cluster. The data reference times mean that the number of times for required data is read or written to the storage units, such as the IRF unit, the SPRF unit, and the off-cluster memory during the program execution. The total number of times for each dedicated off-cluster memory of the ALU_0 unit, ALU_1 unit, MUL_0 unit, MUL_1 unit, and DIV_0 unit to be read/written during executing the FIR filter simulation are 0/21, 0/0, 0/2, 0/0, 16/1, 10/0, 16/1, 10/0, 0/0, and 0/0, respectively. In addition, the total number of times for the SPRF unit and each dedicated IRF unit of the ALU_0 unit, ALU_1 unit, MUL_0 unit, MUL_1 unit, and DIV_0 unit to be read/written during executing the FIR filter simulation are 7/7, 57/57, 56/56, 75/75, 75/75, 80/16, 80/10, 80/16, 08/10, 0/0, and 0/0, respectively.

From the results of performance evaluation described in the previous paragraph, the proportion of data reference rate between the on-cluster memory, such as the IRF units and SPRF unit, and off-cluster memory, such as the SRAM, is 912 : 77. Furthermore, a proportion is 0 : 885 if there is no the hierarchy memory bandwidth; in other words, without containing any IRF units and SPRF unit in the ALU Cluster. The more times of data reference to off-chip memory, the more latency and execution time to finish the program simulation. Therefore, this has proven that the hierarchy memory bandwidth could be used efficiently and effectively in the sense that expensive and communication limited global memory bandwidth is not wasted on the arithmetic units where inexpensive local memory bandwidth is easy to use and provide.

Moreover, for the proposed media streaming processor with three-level hierarchy memory bandwidth architecture, the proportion of data reference rate could be inferred as 912 : 128 : 51 from the on-chip memory to off-chip memory, for instance, from the ALU Cluster, stream register file, to DDR RAM, while executing the FIR filter simulation. This could further demonstrate the hierarchy memory bandwidth to be utilized well and significantly.

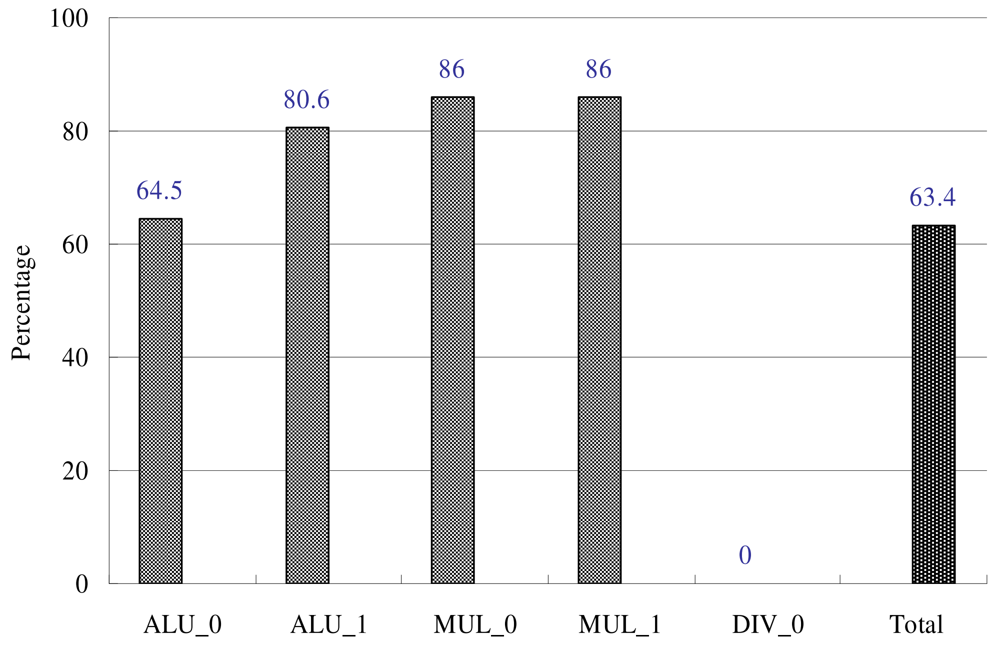


Figure 4.3.3.1: The Code Utilization of Each Arithmetic Unit

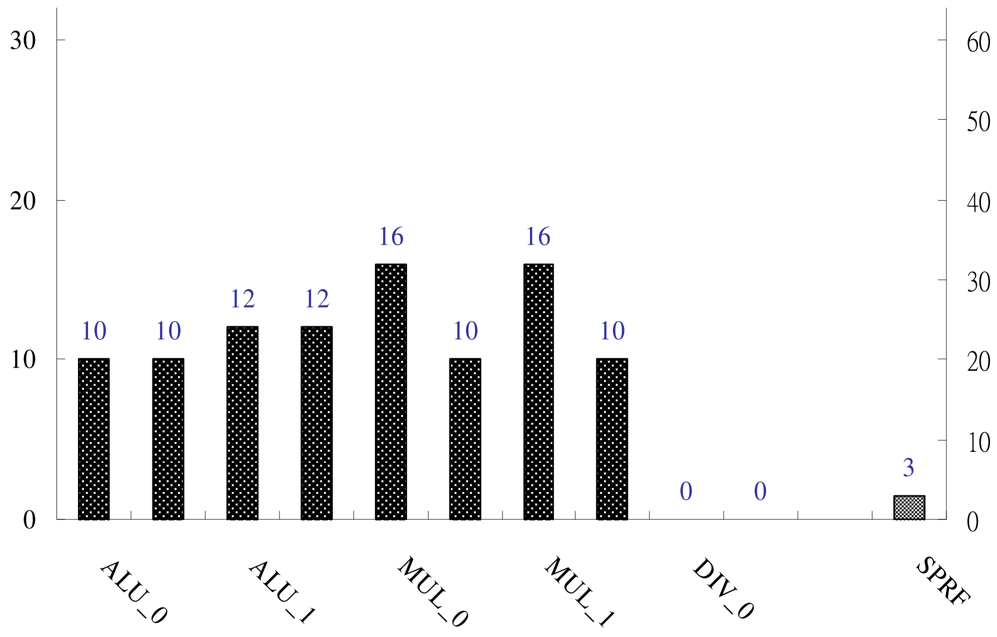


Figure 4.3.3.2: The Memory Utilization for Capacity Usage

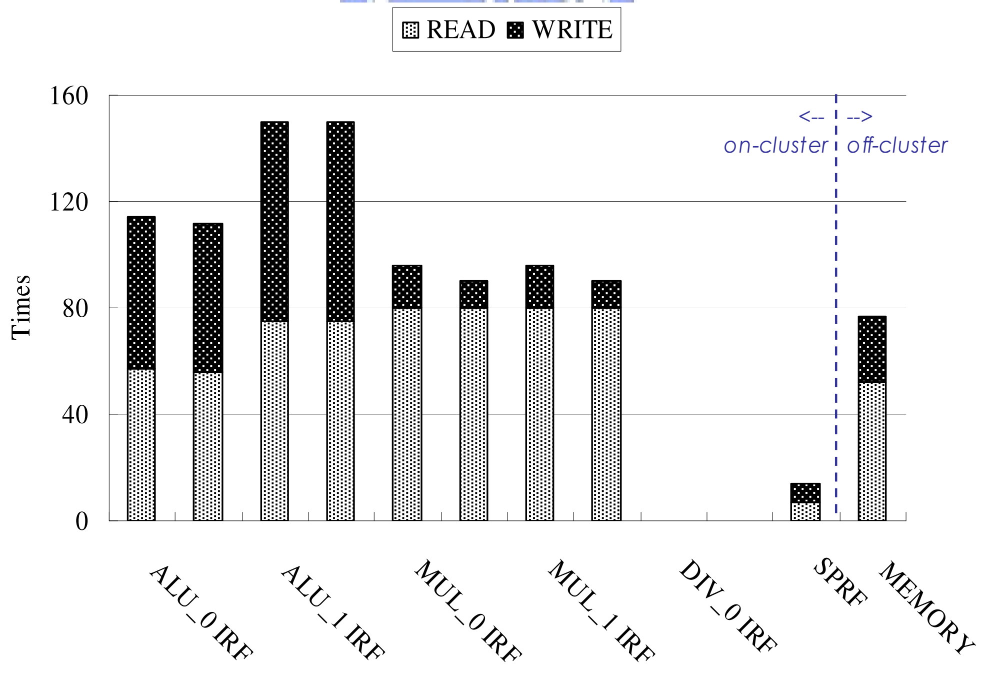


Figure 4.3.3.3: The Memory Utilization for Data Reference

4.4 Performance Comparison

In order to estimate that whether the performance evaluation of this work after having executed the simulation of FIR filter own the competitiveness or not, this ALU Cluster is compared to recent relative reported works in three different design architecture styles: the reconfigurable architecture, the application-specific architecture, and the field programmable gate array (FPGA) architecture. This work and Schmit [28] are related to the reconfigurable architectures. Stefatos [29], Wang [30], and Staszewski [31] are related to the application-specific architectures. Finally, Atmel AT6000 [32] is related to the FPGA architecture. The detail comparison results of these works are listed in Table 4.4.1.

Table 4.4.1: Comparison Results

Paper	*Process*	*Size*	*Freq (MHz)*	*Area (mm^2)*	*Power (mW)*	*Architecture*
This Work (2005)	UMC 0.18 um	32-bit (16-tap)	90.9	1.47	312.38	reconfigurable
Schmit (2002)	ST 0.18 um	32-bit (16-tap)	120	55.48	650	
Stefatos (2005)	UMC 0.18 um	20-bit 32-tap	100	1.465	181	application-specific
Wang (2005)	0.18 um	16-bit 73-tap	10	0.601	8.839	
Staszewski (2000)	TI 0.18 um	6-bit 8-tap	550	0.3	36	
Atmel AT6000	0.6 um	20-bit 16-tap	76.9	1280	496	FPGA

Figure 4.4.1 is shown a chart that those reported works in Table 4.4.1 is normalized to this work. For operation frequency, the upward column means that clock rate is faster than this work, and the downward column means that clock rate is slower than this work. Similarly, for physical area and power dissipation, the downward column means that die size and power consumption are fewer than this work correspondingly, and the upward column means that die size and power consumption are larger than this work correspondingly.

In a word, this work could be performed a quite competitive performance while being compared with the application-specific architectures by trading the larger physical area and the more power dissipation for the faster operation frequency. However, the application range of this work is more widely and flexibly than the application-specific architectures to suitable for handling various media processing applications. Besides, this work has better performance in the operation frequency, physical area, and power dissipation that are compared with the reconfigurable architectures and the FPGA architectures, respectively. Hence, this work has provided a breakthrough in the operating time, die size, and power saving among these general-purpose architectures.

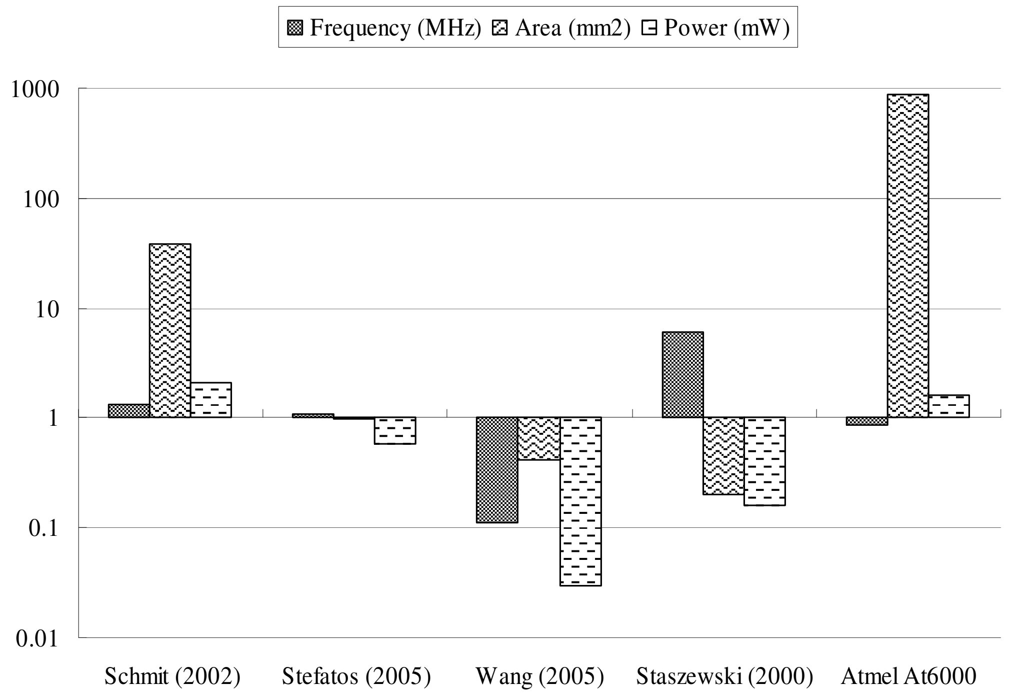


Figure 4.4.1: The Normalized Comparison Results

4.5 Low Power Techniques Implementation

The design of ALU Cluster combined with power saving techniques has been developed and implemented at SoC Laboratory by students and faculty [33]. The prototype of low power media processor architecture is shown in Figure 4.5.1. There are two low power techniques utilized in this work: power gating [34] and voltage islands [35]. Power gating, or is also called sleep transistor, is commonly used to disconnect the power supply of function block when it is in the idle mode. This is achieved by connecting a transistor in series with the power supply of function block. Sleep transistor technique involves to partition the chip into different blocks depended on the functionality that might be selectively powered on or off. When the function block is in the sleep state, the sleep transistor is turned off. Thus, the power dissipation of this function block could be reduced. In practice, a network of sleep transistor might be necessary to efficiently control and decrease the leakage power dissipation. Function blocks which might be periodically powered off are isolated from the primary power distribution network of chip by placing them into voltage islands.

The cell-based design flow is utilized to finish this design, and in the mean time the EDA flow and CAD tools are investigated in order to provide the low power circuitry controlling techniques embedded. Hence, the layout of four ALU Clusters with power saving equipments is shown in Figure 4.5.2. Each ALU Cluster surrounded by its own power ring that is governed by the control logic.

Besides, based on the developed architecture simulator, it could determine how many ALU Clusters to be executed to reach the best performance that trades between the execution time and power dissipation after having simulated an assigned media processing application. Here the FFT is executed in the multi-cluster architecture. No matter how many numbers of the ALU Cluster are used on the multi-cluster architecture to simulate FFT, the power dissipation would almost be the same if there is no any low power technique design embedded. However, as shown in Figure 4.5.3, by utilizing the low power technique design, the power dissipation is scalable and decreases 62% for using one ALU Cluster. In addition, if more ALU Clusters are used on the multi-cluster architecture, then running a program execution could be completed more rapidly, and the total energy consumption could be also decreased since the execution time could be largely reduced. As shown in Figure 4.5.4, the energy consumption decreases 33% while using the numbers of four times of ALU Cluster. Furthermore, if the system has to lower the power dissipation, then only use a portion of ALU Cluster and turn of the idle ALU Cluster. Although this would increase more execution time, the power dissipation could be greatly decreased.

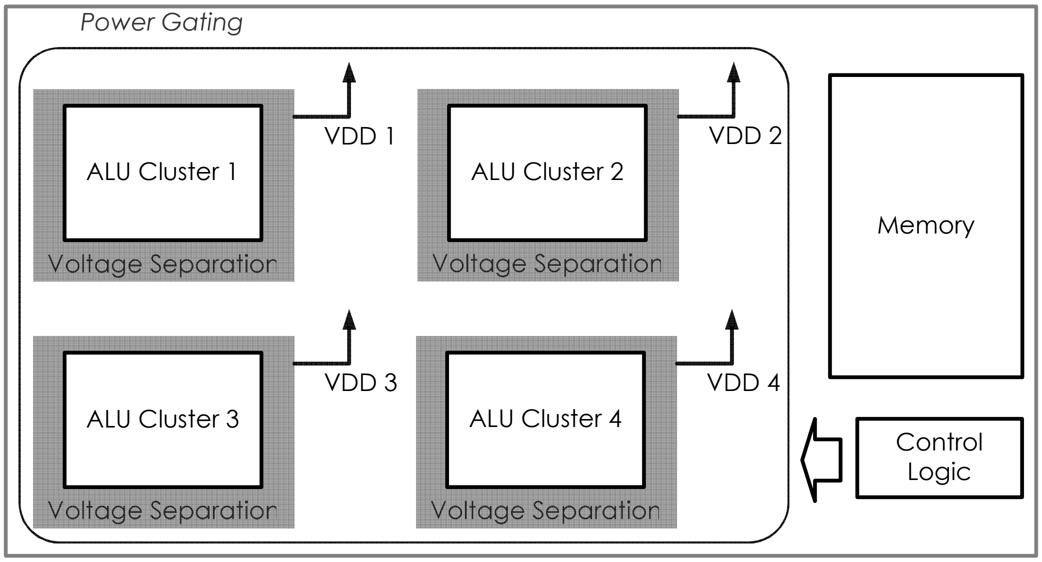


Figure 4.5.1: Multi-Cluster Architecture with Low Power Techniques

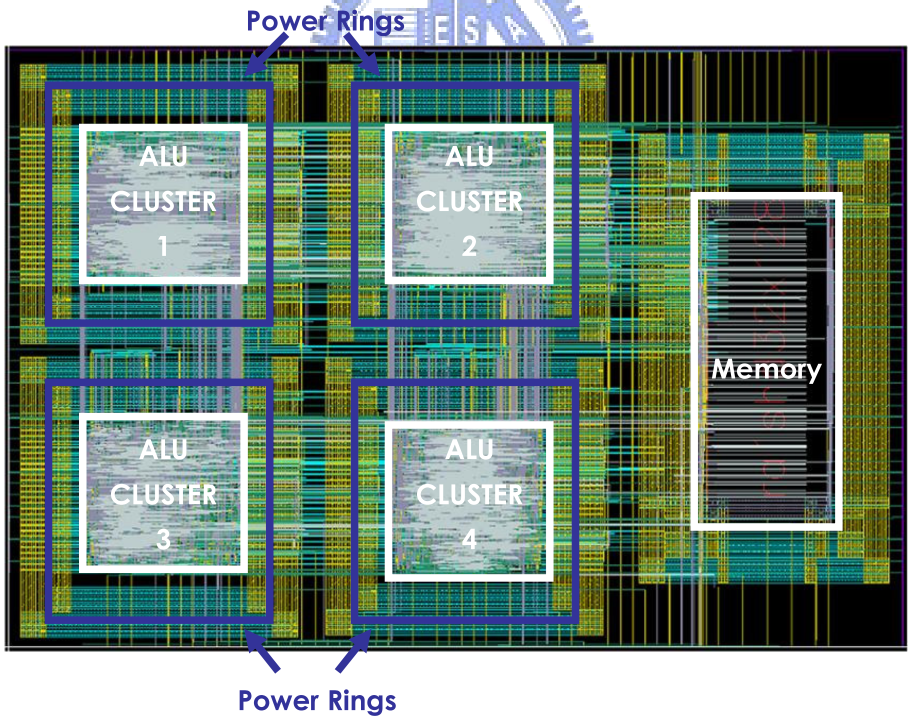


Figure 4.5.2: Layout of Multi-Cluster with Low Power Equipments

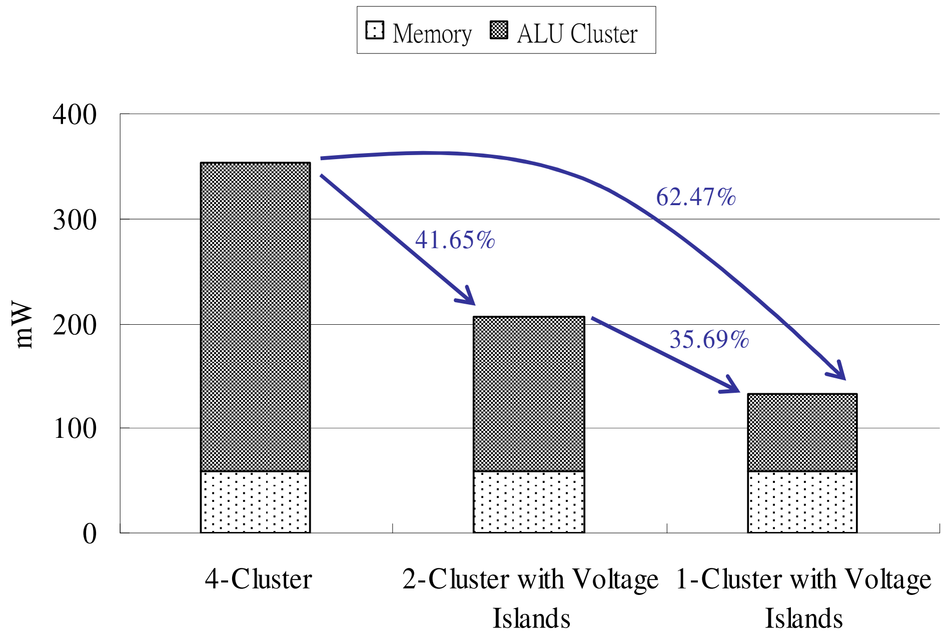


Figure 4.5.3: Power Dissipation on the Multi-Cluster Architecture

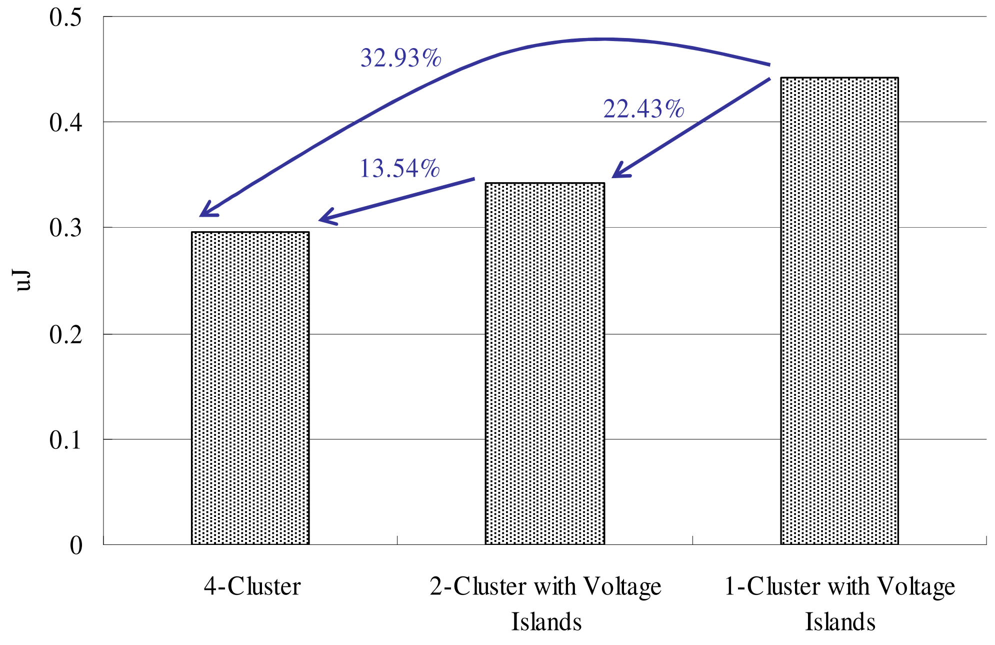


Figure 4.5.4: Energy Consumption on the Multi-Cluster Architecture

To integrate the developed ALU Cluster with power saving techniques, the execution time and power dissipation depended on the performance requirement would be scalable and well fit for this work. From the above-mentioned results, therefore, the combination of streaming architectures and power saving techniques would be the main stream for the design of next generation portable multimedia and communication systems in the future as depicted in Figure 4.5.5.

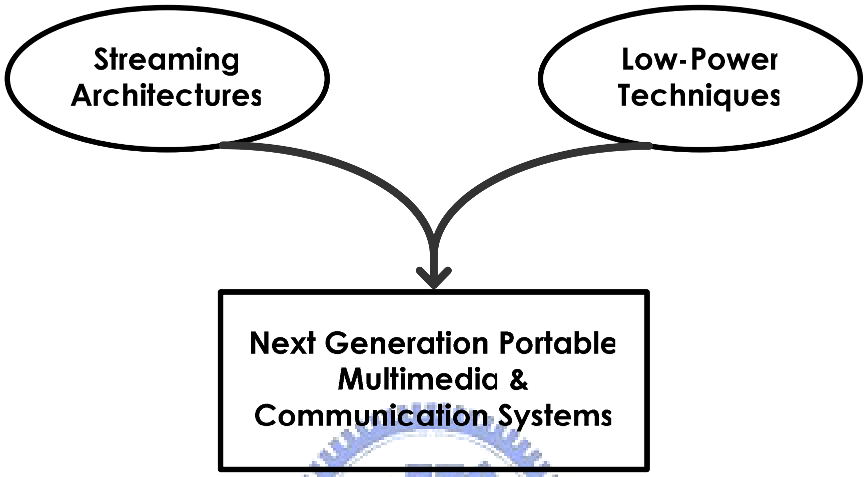


Figure 4.5.5: Future Mobile Multimedia and Communication System Design Trend

4.6 Summary

We have fully implemented the ALU Cluster architecture design by utilizing the current developed CAD tools and cell-based design flow. The maximal clock rate, physical core size, and power dissipation are 100 MHz, 2.16 mm^2, and 312 mW, respectively. After having completed the execution of selected benchmark simulation, FIR filter system, the code utilization is 63.4%, and the clock cycles per executed result output is 3.96. The memory capacity of IRF units and SPRF unit are sufficient to provide during the simulation execution, and the ratio of data reference times of on-cluster memory and off-cluster memory is 989 : 91. The higher ratio to the on-cluster memory means that the limited global memory bandwidth is not wasted on the arithmetic units where the ample local memory bandwidth is easy to utilize. Compare to current related reported works, this work could perform a quite competitive performance while being compared with the application-specific architectures. Besides, this work has better performance compared with the reconfigurable architectures and the FPGA architectures.

Furthermore, the implementation of multiple ALU Clusters design combined with power saving techniques has been developed by several members of SoC Laboratory, and another developed architecture simulator could depend on the required performance to determine the number of ALU Clusters to be executed after the application simulation has been finished. Therefore, the performance of execution time and power dissipation would be scalable depended on requirement.

CHAPTER 5
CONCLUSION

An ALU Cluster design for the media streaming processors architecture has been designed in this thesis. In the meantime, this work has also demonstrated the consideration of implementation feasibility of each component. The back-end simulation results based on the process technology and standard cell library have decided the optimized number and performance of each component. This streaming architecture combined with memory bandwidth hierarchy architecture has efficiently dealt with the selected test bench without wasting too much expensive and communication limited global memory bandwidth on the function units. Additionally, the analysis results of performance evaluation for this work confirm to have the competitiveness and advantages compared with recent relative reported works. Finally, the prototype of this work has been fabricated in UMC 0.18 um 1P6M standard CMOS process technology.

To integrated the developed ALU Cluster with power saving techniques as shown in Figure 4.5.1. The results show that the power dissipation and energy consumption of selected benchmark for the multimedia applications and baseband communication systems could be reduced significantly. Both power dissipation and energy consumption become scalable by dynamic selecting the number of utilized ALU Clusters. The instant performance and energy consumption of an entire work could be optimized for mobile systems. Thus, this design has provided a breakthrough in the operating time and power dissipation in limited battery life for similar architectures. From the above-mentioned results, therefore, the combination of streaming architectures and power saving techniques have been the main stream for the design of next generation portable multimedia and communication systems as depicted in Figure 4.5.5.

BIBLIOGRAPHY

[1] S. Rixner, W. J. Dally, U. J. Kapasi, B. Khailany, A. Lopez-Lagunas, P. R. Mattson, J. D. Owens, "A Bandwidth-Efficient Architecture for Media Processing," *Proceedings of 31st Annual ACM/IEEE International Symposium on Microarchitecture*, pages 3-13, November 1998.

[2] L. Hennessy, A. Patterson, *Computer Architecture: A Quantitative Approach*, Third Edition, Morgan Kaufmann Publishers, 2003.

[3] W. Wolf, *Modern VLSI Design: System-on-Chip Design*, Third Edition, Prentice Hall Modern Semiconductor Design Series, 2002.

[4] J. Rabaey, A. Chandrakasan, B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, Second Edition, Prentice Hall Electronics and VLSI Series, 2003.

[5] N. H. E. Weste, K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, Second Edition, Addison-Wesley VLSI Systems Series, 1993.

[6] 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.

[7] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Matttson, J. D. Owens, "Programmable Stream Processors," *IEEE Computer*, pages 54-62, August 2003.

[8] 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.

[9] 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.

[10] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, I. Kim, G. Daglikoca, "The Architecture of the DIVA Processing-In-Memory Chip," *Proceedings of the International Conference on Supercomputing*, pages 14-25, June 2002.

[11] J. Draper, J. Sondeen, S. Mediratta, I. Kim, "Implementation of a 32-bit RISC Processor for the Data-Intensive Architecture Processing-In-Memory Chip," *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pages 163-172, July 2002.

[12] T. Sakurai, "Perspectives on Power-Aware Electronics," *IEEE International Solid-State Circuits Conference*, pages 26-29, February 2003.

[13] J. Mitola, "The Software Radio Architecture," *IEEE Communications Magazine*, pages 26-38, May 1995.

[14] E. Buracchini, "The Software Radio Concept," *IEEE Communications Magazine*, pages 138-143, September 2000.

[15] M. Keating, P. Bricaud, *Reuse Methodology Manual for System-on-Chip Designs*, Third Edition, Kluwer Academic Publishers, 2002.

[16] J. D. Owens, S. Rixner, U. J. Kapasi, P. Mattson, B. Towles, B. Serebrin, W. J. Dally, "Media Processing Applications on the Imagine Stream Processor," *Proceedings of the IEEE International Conference on Computer Design*, pages 295-302, September 2002.

[17] A. V. Oppenheim, R. W. Schafer, J. R. Buck, *Discrete-Time Signal Processing*, Second Edition, Prentice Hall Signal Processing Series, 1999.

[18] A. P. Chandrakasan, S. Sheng, R. W. Brodersen, "Low-Power CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, pages 473-484, April 1992.

[19] S. Rixner, *Stream Processor Architecture*, Kluwer Academic Publishers, 2002.

[20] B. khailany, *The VLSI Implementation and Evaluation of Area- and Energy-Efficient Streaming Media Processors*, Ph.D Dissertation, Stanford University, 2003.

[21] http://www.synopsys.com/

[22] http://www.umc.com/english/process/d.asp
http://www.umc.com/english/design/b_3.asp#Artisan

[23] R. Ho, K. Mai, M. Horowitz, "The Future of Wires," *Proceedings of the IEEE*, pages 490-504, April 2001.

[24] http://www.cadence.com/

[25] http://www.novas.com/

[26] http://www.mentor.com/

[27] http://www.mathworks.com/

[28] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, R. R. Taylor, "PipeRench: A Virtualized Programmable Datapath in 0.18 Micron Technology," *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 63-66, May 2002.

[29] E. F. Stefatos, H. Wei, T. Arslan, R. Thomson, "Low-Power Reconfigurable VLSI Architecture for the Implementation of FIR Filters," *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, pages 168b-168b, April 2005.

[30] C. H. Wang, A. T. Erdogan, T. Arslan, "Algorithmic Implementation of Low-Power High Performance FIR Filtering IP Cores," *Proceedings of the 18th IEEE International Conference on VLSI Design*, pages 659-662, January 2005.

[31] R. B. Staszewski, K. Muhammad, P. Balsara, "A 550-MSample/s 8-Tap FIR Digital Filter for Magnetic Recording Read Channels," *IEEE Journal of Solid-State Circuits*, pages 1205-1210, August 2000.

[32] http://www.atmel.com/dyn/resources/prod_documents/DOC0833.PDF

[33] T. W. Lin, M. C. Lee, F. J. Lin, H. Chiueh, "A Low Power ALU Cluster Design for Media Streaming Architecture," *to appear: IEEE 48th International Midwest Symposium on Circuits and Systems*, August 2005.

[34] J. W. Tschanz, S. g. Narendra, Y. Ye, B. A. Bloechel, s. Borkar, V. De, "Dynamic Sleep Transistor and Body Bias for Active Leakage Power Control of Microprocessors," *IEEE Journal of Solid-State Circuits*, pages 1838-1845, November 2003.

[35] D. E. Lackey, P. S. Zuchowski, T. R. Bednar, D. W. Stout, S. W. Gould, J. M. Cohn, "Managing Power and Performance for System-on-Chip Design Using Voltage Islands," *IEEE/ACM International Conference on Computer Aided Design*, pages 195-202, November 2002.

APPENDIX A
SUMMARY OF THE DEFINED MICROCODE IN INSTRUCTION SET

I. The part of "SOURCE" in instruction set format

Source Register	Binary Code [1:0]
None operation	00 $_b$
Off-chip data memory (DM)	11 $_b$
SPRF (SP)	10 $_b$
IRF (RF)	11 $_b$

II. The part of "DESTINATION" in instruction set format

Destination Register	Binary Code [3:0]
None operation	0000 $_b$
Off-chip data memory (DM)	0001 $_b$
SPRF (SP)	0010 $_b$
Left IRF of ALU_0 (I9)	0011 $_b$
Right IRF of ALU_0 (I8)	0100 $_b$
Left IRF of ALU_1 (I7)	0101 $_b$
Right IRF of ALU_1 (I6)	0110 $_b$
Left IRF of MUL_0 (I5)	0111 $_b$
Right IRF of MUL_0 (I4)	1000 $_b$
Left IRF of MUL_1 (I3)	1001 $_b$
Right IRF of MUL_1 (I2)	1010 $_b$
Left IRF of DIV_0 (I1)	1011 $_b$
Right IRF of DIV_0 (I0)	1100 $_b$

III. The part of “OPERATION CODE” in instruction set format

Function Unit	Operation	OP code
ALU	None	0000_b
	ADD	0001_b
	SUB	0010_b
	ABS	0011_b
	AND	0100_b
	OR	0101_b
	XOR	0110_b
	NOT	0111_b
	SLL	1000_b
	SRL	1001_b
	SRA	1010_b
	LT	1011_b
	GT	1100_b
	EQ	1101_b
MUL	None	0_b
	MUL	1_b
DIV	None	00_b
	DIV	01_b
	REM	10_b
	SQR	11_b

APPENDIX B

ASSEMBLY CODE OF TEST BENCH

#	ALU_0	ALU_1	MUL_0	MUL_1	DIV_0
1			DM_16_DM_01_DM_20_MUL	DM_15_DM_01_I9_00_MUL	
2			DM_16_DM_02_I8_00_MUL	DM_14_DM_01_I7_00_MUL	
3			DM_15_DM_02_I6_00_MUL	DM_16_DM_03_I9_01_MUL	
4			DM_13_DM_01_I7_01_MUL	DM_14_DM_02_I6_01_MUL	
5			DM_15_DM_03_I7_02_MUL	DM_16_DM_04_I6_02_MUL	
6			DM_12_DM_01_I7_03_MUL	DM_13_DM_02_I6_03_MUL	
7			DM_14_DM_03_I7_04_MUL	DM_15_DM_04_I6_04_MUL	
8			DM_16_DM_05_I8_01_MUL	DM_11_DM_01_I7_05_MUL	
9	RF_00_RF_00_DM_00_ADD		DM_12_DM_02_I6_05_MUL	DM_13_DM_03_I7_06_MUL	
10		RF_00_RF_00_SP_00_ADD	DM_14_DM_04_I6_06_MUL	DM_15_DM_05_I7_07_MUL	
11		RF_01_RF_01_I9_02_ADD	DM_16_DM_06_I6_07_MUL	DM_10_DM_01_I7_08_MUL	
12		RF_02_RF_02_I8_02_ADD	DM_11_DM_02_I6_08_MUL	DM_12_DM_03_I7_09_MUL	
13		RF_03_RF_03_I9_03_ADD	DM_13_DM_04_I6_09_MUL	DM_14_DM_05_I7_10_MUL	
14		RF_04_RF_04_I8_00_ADD	DM_15_DM_06_I6_10_MUL	DM_16_DM_07_I9_00_MUL	
15	RF_01_SP_00_DM_01_ADD		DM_09_DM_01_I7_00_MUL	DM_10_DM_02_I6_00_MUL	
16		RF_05_RF_05_I8_03_ADD	DM_11_DM_03_I7_01_MUL	DM_12_DM_04_I6_01_MUL	
17	RF_02_RF_02_DM_02_ADD	RF_06_RF_06_I9_04_ADD	DM_13_DM_05_I7_02_MUL	DM_14_DM_06_I6_02_MUL	
18	RF_03_RF_01_I9_05_ADD	RF_07_RF_07_I8_04_ADD	DM_15_DM_07_I7_03_MUL	DM_16_DM_08_I6_03_MUL	
19		RF_08_RF_08_I8_05_ADD	DM_08_DM_01_I7_04_MUL	DM_09_DM_02_I6_04_MUL	
20		RF_09_RF_09_I9_01_ADD	DM_10_DM_03_I7_11_MUL	DM_11_DM_04_I6_11_MUL	
21		RF_10_RF_10_I8_06_ADD	DM_12_DM_05_I7_05_MUL	DM_13_DM_06_I6_05_MUL	
22	RF_04_RF_03_I9_06_ADD	RF_00_RF_00_I8_07_ADD	DM_14_DM_07_I7_06_MUL	DM_15_DM_08_I6_06_MUL	
23	RF_05_RF_00_DM_03_ADD	RF_01_RF_01_I9_02_ADD	DM_16_DM_09_I8_01_MUL	DM_07_DM_01_I7_07_MUL	
24	RF_00_RF_05_I9_03_ADD	RF_02_RF_02_I8_02_ADD	DM_08_DM_02_I6_07_MUL	DM_09_DM_03_I7_08_MUL	
25		RF_03_RF_03_I9_07_ADD	DM_10_DM_04_I6_08_MUL	DM_11_DM_05_I7_09_MUL	
26	RF_01_RF_06_I8_08_ADD	RF_04_RF_04_I9_08_ADD	DM_12_DM_06_I6_09_MUL	DM_13_DM_07_I7_10_MUL	
27	RF_06_RF_04_DM_04_ADD	RF_11_RF_11_I8_03_ADD	DM_14_DM_08_I6_10_MUL	DM_15_DM_09_I7_00_MUL	
28	RF_02_RF_07_I8_00_ADD	RF_05_RF_05_I9_04_ADD	DM_16_DM_10_I6_00_MUL	DM_06_DM_01_I7_01_MUL	

29		RF_06_RF_06_I8_05_ADD	DM_07_DM_02_I6_01_MUL	DM_08_DM_03_I7_02_MUL	
30	RF_07_RF_02_I9_00_ADD		DM_09_DM_04_I6_02_MUL	DM_10_DM_05_I7_03_MUL	
31	RF_08_RF_01_I9_01_ADD	RF_07_RF_07_I8_06_ADD	DM_11_DM_06_I6_03_MUL	DM_12_DM_07_I7_04_MUL	
32	RF_03_RF_08_DM_05_ADD	RF_08_RF_08_I9_05_ADD	DM_13_DM_08_I6_04_MUL	DM_14_DM_09_I7_11_MUL	
33	RF_04_RF_03_I9_02_ADD	RF_09_RF_09_I8_04_ADD	DM_15_DM_10_I6_11_MUL	DM_05_DM_01_I7_05_MUL	
34		RF_10_RF_10_I9_06_ADD	DM_06_DM_02_I6_05_MUL	DM_07_DM_03_I7_06_MUL	
35	RF_00_RF_00_DM_06_ADD	RF_00_RF_00_I8_02_ADD	DM_08_DM_04_I6_06_MUL	DM_09_DM_05_I7_12_MUL	
36	RF_01_RF_05_I8_01_ADD	RF_01_RF_01_I9_07_ADD	DM_10_DM_06_I6_12_MUL	DM_11_DM_07_I7_07_MUL	
37	RF_05_RF_06_I9_03_ADD	RF_02_RF_02_I8_07_ADD	DM_12_DM_08_I6_07_MUL	DM_13_DM_09_I7_08_MUL	
38		RF_03_RF_03_I9_04_ADD	DM_14_DM_10_I6_08_MUL	DM_04_DM_01_I7_09_MUL	
39	RF_06_RF_04_I9_08_ADD	RF_04_RF_04_I8_03_ADD	DM_05_DM_02_I6_09_MUL	DM_06_DM_03_I7_10_MUL	
40		RF_11_RF_11_I8_00_ADD	DM_07_DM_04_I6_10_MUL	DM_08_DM_05_I7_00_MUL	
41	RF_02_RF_01_DM_07_ADD	RF_05_RF_05_I9_00_ADD	DM_09_DM_06_I6_00_MUL	DM_10_DM_07_I7_01_MUL	
42	RF_03_RF_02_SP_00_ADD	RF_06_RF_06_I8_05_ADD	DM_11_DM_08_I6_01_MUL	DM_12_DM_09_I7_02_MUL	
43	RF_07_RF_07_I9_01_ADD	RF_12_RF_12_I8_06_ADD	DM_13_DM_10_I6_02_MUL	DM_03_DM_01_I7_03_MUL	
44	RF_04_RF_03_I8_04_ADD	RF_07_RF_07_I9_05_ADD	DM_04_DM_02_I6_03_MUL	DM_05_DM_03_I7_04_MUL	
45		RF_08_RF_08_I9_06_ADD	DM_06_DM_04_I6_04_MUL	DM_07_DM_05_I7_11_MUL	
46		RF_09_RF_09_I9_02_ADD	DM_08_DM_06_I6_11_MUL	DM_09_DM_07_I7_05_MUL	
47	RF_08_SP_00_DM_08_ADD	RF_10_RF_10_I8_01_ADD	DM_10_DM_08_I6_05_MUL	DM_11_DM_09_I7_06_MUL	
48	RF_01_RF_00_I9_03_ADD	RF_00_RF_00_I8_02_ADD	DM_12_DM_10_I6_06_MUL	DM_02_DM_01_I7_12_MUL	
49	RF_00_RF_05_I8_03_ADD	RF_01_RF_01_I9_04_ADD	DM_03_DM_02_I6_12_MUL	DM_04_DM_03_I7_07_MUL	
50	RF_05_RF_06_I8_07_ADD	RF_02_RF_02_I9_07_ADD	DM_05_DM_04_I6_07_MUL	DM_06_DM_05_I7_08_MUL	
51		RF_03_RF_03_I9_09_ADD	DM_07_DM_06_I6_08_MUL	DM_08_DM_07_I7_09_MUL	
52	RF_02_RF_01_I9_08_ADD	RF_04_RF_04_I8_08_ADD	DM_09_DM_08_I6_09_MUL	DM_10_DM_09_I7_10_MUL	
53	RF_03_RF_04_DM_09_ADD	RF_11_RF_11_I9_01_ADD	DM_11_DM_10_I6_10_MUL	DM_01_DM_01_I7_00_MUL	
54	RF_06_RF_03_I9_00_ADD	RF_05_RF_05_I8_00_ADD	DM_02_DM_02_I6_00_MUL	DM_03_DM_03_I7_01_MUL	
55	RF_04_RF_02_I8_05_ADD	RF_06_RF_06_I9_05_ADD	DM_04_DM_04_I6_01_MUL	DM_05_DM_05_I7_02_MUL	
56		RF_12_RF_12_I8_06_ADD	DM_06_DM_06_I6_02_MUL	DM_07_DM_07_I7_03_MUL	
57	RF_09_RF_08_I8_01_ADD	RF_07_RF_07_I9_02_ADD	DM_08_DM_08_I6_03_MUL	DM_09_DM_09_I7_04_MUL	
58		RF_08_RF_08_I8_04_ADD	DM_10_DM_10_I6_04_MUL	DM_01_DM_02_I7_11_MUL	
59	RF_00_RF_07_DM_10_ADD	RF_09_RF_09_I9_03_ADD	DM_02_DM_03_I6_11_MUL	DM_03_DM_04_I7_05_MUL	
60	RF_07_RF_05_I8_02_ADD	RF_10_RF_10_I9_04_ADD	DM_04_DM_05_I6_05_MUL	DM_05_DM_06_I7_06_MUL	
61	RF_01_RF_00_I8_03_ADD	RF_00_RF_00_I9_06_ADD	DM_06_DM_07_I6_06_MUL	DM_07_DM_08_I7_12_MUL	
62	RF_05_RF_01_I9_09_ADD	RF_01_RF_01_I8_09_ADD	DM_08_DM_09_I6_12_MUL	DM_09_DM_10_I8_08_MUL	
63	RF_02_RF_06_I8_10_ADD	RF_02_RF_02_I9_10_ADD	DM_01_DM_03_I7_07_MUL	DM_02_DM_04_I6_07_MUL	
64	RF_03_RF_04_I9_00_ADD	RF_03_RF_03_SP_00_ADD	DM_03_DM_05_I7_08_MUL	DM_04_DM_06_I6_08_MUL	
65	RF_08_RF_02_DM_11_ADD	RF_04_RF_04_I9_07_ADD	DM_05_DM_07_I7_09_MUL	DM_06_DM_08_I6_09_MUL	

66		RF_11_RF_11_I8_00_ADD	DM_01_DM_04_I7_10_MUL	DM_02_DM_05_I6_10_MUL	
67	RF_09_RF_03_DM_12_ADD	RF_05_RF_05_I9_01_ADD	DM_03_DM_06_I7_00_MUL	DM_04_DM_07_I6_00_MUL	
68	RF_04_RF_10_I8_01_ADD	RF_06_RF_06_I9_02_ADD	DM_05_DM_08_I7_01_MUL	DM_06_DM_09_I6_01_MUL	
69	RF_06_RF_09_I8_04_ADD	RF_12_RF_12_SP_01_ADD	DM_07_DM_10_I9_03_MUL	DM_01_DM_05_I7_02_MUL	
70	RF_10_SP_00_I9_05_ADD	RF_07_RF_07_I8_02_ADD	DM_02_DM_06_I6_02_MUL	DM_03_DM_07_I7_03_MUL	
71		RF_08_RF_08_SP_02_ADD	DM_04_DM_08_I6_03_MUL	DM_05_DM_09_I7_04_MUL	
72	RF_01_RF_00_I9_08_ADD	RF_09_RF_09_I8_03_ADD	DM_06_DM_10_I6_04_MUL	DM_01_DM_06_I7_05_MUL	
73	RF_00_RF_01_DM_13_ADD	RF_10_RF_10_I8_05_ADD	DM_02_DM_07_I6_05_MUL	DM_03_DM_08_I7_06_MUL	
74	RF_07_RF_04_I8_07_ADD	RF_00_RF_00_I9_04_ADD	DM_04_DM_09_I6_06_MUL	DM_05_DM_10_I8_06_MUL	
75	RF_02_SP_01_I9_06_ADD	RF_01_RF_01_I8_09_ADD	DM_01_DM_07_I7_07_MUL	DM_02_DM_08_I6_07_MUL	
76	SP_02_RF_02_I9_09_ADD		DM_03_DM_09_I7_08_MUL	DM_04_DM_10_I6_08_MUL	
77		RF_02_RF_02_I8_00_ADD	DM_01_DM_08_I7_09_MUL	DM_02_DM_09_I6_09_MUL	
78	RF_03_RF_05_I8_01_ADD	RF_03_RF_03_I9_01_ADD	DM_03_DM_10_I6_11_MUL	DM_01_DM_09_I7_10_MUL	
79	RF_05_RF_07_DM_14_ADD	RF_04_RF_04_I8_04_ADD	DM_02_DM_10_I6_10_MUL	DM_01_DM_10_DM_20_MUL	
80	RF_08_RF_08_I8_10_ADD	RF_05_RF_05_SP_00_ADD	DM_07_DM_09_I7_30_MUL	DM_08_DM_10_I6_30_MUL	
81	RF_04_RF_09_I9_02_ADD	RF_06_RF_06_I8_02_ADD			
82	RF_09_RF_03_SP_63_ADD	RF_07_RF_07_I9_07_ADD			
83	RF_01_RF_00_I9_03_ADD	RF_08_RF_08_I8_05_ADD			
84		RF_09_RF_09_I7_00_ADD			
85	RF_06_RF_10_DM_15_ADD				
86	SP_00_RF_06_I9_00_ADD				
87		RF_10_RF_10_DM_01_ADD			
88	RF_03_RF_04_DM_18_ADD	RF_30_RF_30_I8_31_ADD			
89	RF_07_RF_05_DM_20_ADD	RF_00_RF_11_DM_00_ADD			
90	RF_02_RF_01_DM_17_ADD				
91	RF_00_RF_02_DM_19_ADD				
92					
93	SP_63_RF_31_DM_16_ADD				