A Generic Low Power Reconfigurable Distributed Arithmetic Processor

Liu Zhenyu

A thesis submitted for the degree of Doctor of Philosophy.
The University of Edinburgh.
May 2008

To my wife Ni Yuanyuan

and

my parents Liu Ziheng, Piao Xueqin

For their love, affection and support.

Abstract

Higher performance, lower cost, increasingly minimizing integrated circuit components, and higher packaging density of chips are ongoing goals of the microelectronic and computer industry. As these goals are being achieved, however, power consumption and flexibility are increasingly becoming bottlenecks that need to be addressed with the new technology in Very Large-Scale Integrated (VLSI) design.

For modern systems, more energy is required to support the powerful computational capability which accords with the increasing requirements, and these requirements cause the change of standards not only in audio and video broadcasting but also in communication such as wireless connection and network protocols. Powerful flexibility and low consumption are repellent, but their combination in one system is the ultimate goal of designers.

A generic domain-specific low-power reconfigurable processor for the distributed arithmetic algorithm is presented in this dissertation. This domain reconfigurable processor features high efficiency in terms of area, power and delay, which approaches the performance of an ASIC design, while retaining the flexibility of programmable platforms. The architecture not only supports typical distributed arithmetic algorithms which can be found in most still picture compression standards and video conferencing standards, but also offers implementation ability for other distributed arithmetic algorithms found in digital signal processing, telecommunication protocols and automatic control.

In this processor, a simple reconfigurable low power control unit is implemented with good performance in area, power and timing. The generic characteristic of the architecture makes it applicable for any small and medium size finite state machines which can be used as control units to implement complex system behaviour and can be found in almost all engineering disciplines. Furthermore, to map target applications efficiently onto the proposed architecture, a new algorithm is introduced for searching for the best common sharing terms set and it keeps the area and power consumption of the implementation at low level. The software implementation of this algorithm is presented, which can be used not only for the proposed architecture in this dissertation but also for all the implementations with adder-based distributed arithmetic algorithms. In addition, some low power design techniques are applied in the architecture, such as unsymmetrical design style including unsymmetrical interconnection arranging, unsymmetrical PTBs selection and unsymmetrical mapping basic computing units. All these design techniques achieve

extraordinary power consumption saving. It is believed that they can be extended to more low power designs and architectures.

The processor presented in this dissertation can be used to implement complex, high performance distributed arithmetic algorithms for communication and image processing applications with low cost in area and power compared with the traditional methods.

Declaration of originality

I hereby declare that the research recorded in this thesis and the thesis itself was composed and originated entirely by myself in the School of Engineering and Electronics at The University of Edinburgh, except when otherwise stated.

_____

Zhenyu Liu

Acknowledgments

My foremost thank goes to my supervisor, Prof. Tughrul Arslan, for giving me a big chance to go to the door of top level research which leads me to the success of the work in this dissertation. His valuable comments were instrumental in shaping the direction of the research. Prof. Tughrul Arslan built a harmonious academic environment in which I could discuss and exchange ideas with members in the lab friendly. I thank him also for leaving enough academic space for me to cultivate the ability to carry out research independently, which leads me to success.

This work was supported in part by the David Mayes Scholarship. I wish my successful work can be a solid prove for the rightness of setting up this scholarship, which is also the best gift to Dr Davis Mayes.

It was a valuable experience to work in System-Level Integration group. Many thanks go to Dr. Ahmet T. Erdogan for being so patient and helping in sharing with me technical knowledge whenever I needed help. His help and support contributed greatly to my work and publications, which is deeply appreciated. I sincerely thank all of the current and former members of the group for their precious time, helpful advice and friendship over the last four years. Here, I would like to mention just a few in particular: Sami Khawam, Yi Ying, Erfu Yang, Han Wei, Ming-Lang Lin, Zhan Cheng, Mark Muir, Yutian Zhao, Jichuan Zhao. Specially, I would like to thank Miss Yuanyuan Ni and Prof. Lili Wang for their suggestions and help in editing this manuscript.

A big thank also goes to Miss Ning Wei for her company during my time in Edinburgh. I thank her for her caring, patience and encouragement that made my tough life easier and carried me on through the difficult times. Her support and understanding is greatly appreciated.

Lastly, and most importantly, I would like to express my deepest gratitude to my parents for their forever love, unconditional sacrifices, endless encouragement and continual support (financial and moral) throughout my academic years. They are always my loyal, ultimate and strong backing which carried me forward and I'm forever in debt to them.

I would express my sincere gratitude to all of you again. This dissertation is only made possible with the love and support from the various people in my life,

Table of Contents

List of Figures

List of Tables

Glossary / Acronyms

ALP	Adaptive Logic Processor
ALU	Arithmetic Logic Unit
AVC	Advanced Video Coding
ASIC	Application Specific Integrated Circuit
BRASS	Berkeley Reconfigurable Architecture System and Software
CAVLC	Context Adaptive Variable Length Coding
C-Box	Connection Boxes
CDMA	Code Division Multiple Access
CIO	Configurable I/O
CLB	Configurable Logic Block
CMOS	Complementary metal–oxide–semiconductor
CORDIC	COordinate Rotation DIgital Computer
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CSD	Canonic Signed-Digit
DA	Distributed Arithmetic
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DES	Data Encryption Standard
DHT	Discrete Hartley Transform
DIF	Decimation-In-Frequency
DIT	Decimation-In-Time

DoSP	Domain Specific Processor
DMA	Direct Memory Access
DPU	Data Path Unit
DSP	Digital Signal Processor
DVB-H	Digital Video Broadcasting – Handheld
DVB-T/DAB	Digital Video/Audio Broadcasting
DWT	Discrete Wavelet Transform
EDA	Electronic Design Automation
EEPROM	Electrically Erasable Programmable Read-only Memory
EPROM	Erasable Programmable Read-only Memory
FFT	Fast Fourier Transform
FIFO	First In, First Out
FIP	Fixed Instruction Processor
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Arrays
FSM	Finite State Machine
GPP	General Purpose Processor
GPU	Graphics Processing Units
GRM	General Routing Matrix
GSM	Global System for Mobile communications
HDL	Hardware Description Language
JPEG	Joint Photographic Experts Group
IDCT	Inverse Discrete Cosine Transform
IOB	Input/output block
LAN	Local Area Network
LUT	Look-Up-Tables

LSB	Least Significant Bit
MAC	Multiply and Accumulate
MCD	MultiCarrier Demultiplexing
MIMO	Multiple-Input Multiple-Output
MIPS	Microprocessor without Interlocked Pipeline Stages
MP3	MPEG-1 Audio Layer 3
MPEG-4	Moving Picture Experts Group standard 4
MSB	Most Significant Bit
NAPA	National Semiconductor's Adaptive Processing Architecture
NRE	NonRecurring Engineering
OBC	Offset Binary Coding
OFDM	Orthogonal Frequency Division Multiplex
VLSI	Very Large-Scale Integrated
PAL	Programmable Array Logic
PE	Processing Element
PDA	Personal Digital Assistant
PGA	Programmable Gate Array
PLA	Programmable Logic Array
PLD	Programmable Logic Device
PMA	Pipeline Memory Array
PTB	Product-Term Block
QoS	Quality of Service
RAM	Random Access Memory
RAP	Reconfigurable Algorithm Processor
RaPiD	Reconfigurable Pipelined Datapath
RISC	Reduced Instruction Set Computer

ROM	Read Only Memory
RPC	Reconfigurable Pipeline Controller
RTL	Register Transfer Level
S-Box	Switch Boxes
SDA	Serial Distributed Arithmetic
SIMD	Single Instruction, Multiple Data
SP	Simple Profile
SoC	System-on-Chip
SRAM	Static Random Access Memory
SMA	Scratchpad Memory Array
STG	State Transition Graph
STT	State Transition Table
TBT	Toggle Bus Transceiver
TD-SCDMA	Time Division-Synchronous Code Division Multiple Access
VCEG	Video Coding Experts Group
VHDL	Verilog hardware description language
VLSI	Very Large-Scale Integrated
VLIW	Very-Long Instruction-Word
WCDMA	Wideband Code Division Multiple Access
WFTA	Winograd Fourier Transform Algorithm
WLAN	Wireless Local Area Network
XPC	External Memory Controller
XPLA	eXtended Programmable Logic Array
ZIA	Zero-power Interconnect Array

Chapter I
Introduction

I.1 Introduction

Higher performance, lower cost, increasingly minimizing integrated circuit components, and higher packaging density of chips are ongoing goals of the microelectronic and computer industry. As these goals are being achieved, however, power consumption and flexibility are increasingly becoming bottlenecks that need to be addressed with the new technology in Very Large-Scale Integrated (VLSI) design.

Both powerful computing ability and long running time are the key features of the handheld and portable devices such as wireless communication terminals, personal digital assistants (PDAs), laptops, etc. The outstanding system performance requires more energy to support powerful computational capability. However, high power consumption directly shortens the running time of portable devices which, in a sense, directly determines the future of devices in the market. Therefore, the success of low-power techniques not only implies battery life in mobile system will be extended, but also reliability in high-performance systems will be improved.

The powerful processing ability of modern system accords with the increasing requirements which cause the change of standards not only in audio and video broadcasting but also in communication such as wireless connection and network protocols. The frequent updates in media and communication standards raise higher requirements in flexibility to support changes. Generally speaking, the more complex the system is, the more power is consumed. Powerful flexibility and low consumption are repellent, but their combination in one system is the ultimate goal of designers.

Programmable solutions such as Field Programmable Gate Arrays (FPGA) devices have become more popular among the applications in multimedia and communication because of their low design cost and fast time-to-market. However, compared with Application Specific Integrated Circuit (ASIC) solutions, there is a large power and delay overhead for these programmable solutions.

Despite the common notion of FPGA's large power consumption, Jan Rabaey has shown in [1] that, for certain type of digital signal processing applications, the energy efficiency of FPGAs is orders of magnitude better than that of general purpose processors. This observation is shown in Figure I-1. Although ASICs or hardwired solutions provide the best energy-efficiency, their longer design cycles make the time-to-market unacceptable for a business company besides their high design costs.

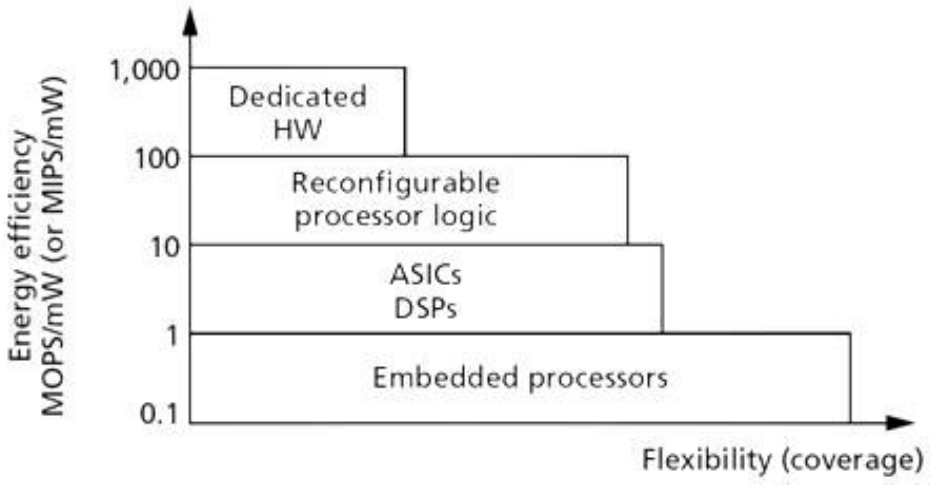

Figure I-1: Flexibility versus energy trade-off in implementation [1]

Reconfigurable System-on-Chip (SoC) technology emerged to meet the simultaneous demands for flexibility and efficiency. Compared with general SoC, one or more programmable arrays are embedded in the reconfigurable system. The reconfigurable arrays can be programmed to adapt to different applications so that the efficiency of the hardware and the flexibility of the whole system are improved. A typical reconfigurable SoC architecture consists of general purpose processor, memory, system bus, control modules and reconfigurable arrays which handle specific complex functions.

From the developing trends in reconfigurable logic and computing, it is found that the reconfigurable computing consumes higher power (roughly a factor of 10) when compared to ASICs. But compared with standard microprocessors the energy-efficiency is about two orders of magnitude better [2], as shown in Figure I-2. Therefore, introduction of reconfigurable architecture can lead to significant energy savings when compared with processor solutions only [2].

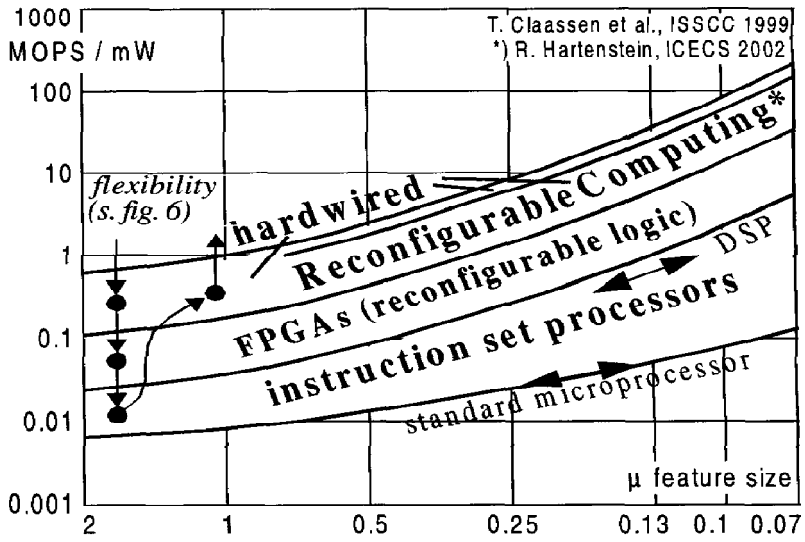

Figure I-2: Energy efficiency vs. flexibility including reconfigurable computing [2]

This thesis presents a hybrid solution between ASICs and general-purpose programmable platforms to fill up this gap. This solution is an application-specific reconfigurable processor targeting on distributed arithmetic algorithm, which approaches the performance of an ASIC design, while retaining the flexibility of programmable platforms.

I.2 Significance of This Work

The contributions of this research are stated as follows, including five key aspects:

- A novel domain-specific reconfigurable architecture for the distributed arithmetic algorithm is demonstrated. The architecture not only supports the typical distributed arithmetic algorithm, discrete cosine transform, which can be found in most still picture compression standards and video conferencing standards, but also offers implementation ability for other distributed arithmetic algorithms such as discrete Fourier transform, finite impulse response, and discrete Hartley transform.

- To achieve the best hardware efficiency, the concepts of dimidiate tree and crossing forest are introduced. A new algorithm is accordingly developed for searching for the best common sharing terms set when the target application is implemented with the proposed architecture. The algorithm can find out the best set for implementation so as to achieve the most efficient consumption of area and power.

- The software implementation of the algorithm for searching for the best common sharing terms set is demonstrated, which can be used not only for the architecture presented in this dissertation but also for all the implementation of adder-based distributed arithmetic algorithm.

- A reconfigurable control unit is introduced, which is not only the key part of proposed architecture. The core part of it, reconfigurable finite state machine architecture, can be applied in any small and medium-sized finite state machines which are the control units to implement complex system behaviour and can be found in almost all engineering disciplines.

- Low power design techniques such as unsymmetrical interconnection arranging, unsymmetrical product-term blocks (PTBs) selection and unsymmetrical mapping basic computing units are presented in this dissertation, which can improve area and power efficiency significantly.

I.3 Organisation of the Thesis

This section describes the organization of this dissertation by introducing main points of each chapter.

Chapter II presents reviews of literature related to this work including basic concepts of reconfigurable architecture, the classification of the architectures, a brief introduction to interconnection network applied in reconfigurable architecture and low power design technology. In addition, the implementations of fine-grain and coarse-grain architectures are described. Typical reconfigurable coarse-grain architectures are also presented and compared in detail.

Chapter III first introduces basic concept of distributed arithmetic algorithm and its definition. Two basic distributed arithmetic algorithms, Read Only Memory (ROM) based distributed arithmetic and adder based distributed arithmetic, are then addressed after the summarization of the distributed arithmetic concept development. Several serial and parallel architectures for distributed arithmetic are described. The advantages of these architectures and the problems facing them are also proposed in this chapter. At the end of this chapter, distributed arithmetic applications in signal processing and communication fields are described briefly.

A domain-specific low power reconfigurable distributed arithmetic architecture and its implementation are addressed in Chapter IV. The overview architecture is described first and then the descriptions of the algorithm logic unit are detailed. To achieve the best hardware efficiency, the concepts of dimidiate tree and crossing forest are introduced and defined in this chapter. An algorithm is accordingly developed and presented in the chapter as well, which makes the architecture mapped with the best efficiency.

The control unit of proposed processor will be described separately in Chapter V because if its complexity and unique function. Based on the analysis of traditional reconfigurable architecture given at the beginning of this chapter, a simplified one is presented with less flexibility, but high efficiency in terms of area, power and delay. In this chapter, the performance of the reconfigurable control unit architecture in area, power and delay of control unit will be evaluated and analyzed as well.

After the full description of the proposed processor in Chapter IV and V, the implementations of two typical distributed arithmetic applications, discrete cosine transform and discrete Fourier transform, are introduced in Chapter VI for functionality verification and performance evaluation. The implementations with the target architecture include the configurations of control path, register matrix and algorithm logic unit specified according to the requirements of the application. Additionally, the common term sharing scheme is demonstrated in this chapter by applying dimidiate tree and the algorithm to search for optimal scheme.

Finally, Chapter 7 concludes the thesis by discussing the contributions of the dissertation, limitations of the proposed architecture and directions for future research.

Chapter II

Review of Reconfigurable and Low-power Architecture

II.1 Overview

The traditional ASIC approach has become very expensive due to large design time and increasing photolithography cost. In addition, the relatively rapid changes in algorithms make an ASIC tend to execute partial reuse of the chip, which has resulted in this approach being widely considered a financially infeasible solution for most applications. This can be overcome by adding flexibility and programmability to ASICs, which allows making changes to the design after fabricating. Thus, design errors are greatly reduced; updated standards are better supported and the system is better able to overcome run-time constraints. Besides, the flexibility helps the system adapt to run-time constraints by adopting dynamic reconfiguration. Currently, such flexibility is realized through software solutions with processors and digital signal processors (DSPs).

However, it is not beneficial in portable devices with performance-critical application such as Moving Picture Experts Group standard 4 (MPEG-4) and Advanced Video Coding (AVC) whose complexity demand high operating frequency and power consumption of DSP to achieve the high throughput required.

Efforts of researchers to find better architectures for future devices have resulted in several novel systems on which the current work presented in this thesis is based. Existing and established architectures like DSPs, FPGAs and ASICs were described previously. Features of typical emerging and reported reconfigurable architectures will be demonstrated in the rest of this chapter. As will be compared later, each architecture has its own pros and cons

and only a few of them can potentially function with high performance and low-power consumption.

This chapter first explores reconfigurable logic structures and reconfigurable computing architectures. Since programmable interconnects contribute greatly to flexibility of reconfigurable systems, a considerable part of this work focuses on the interconnections. The second part of this chapter overviews the existing programmable interconnection topologies. The last part of the chapter describes low power technologies briefly.

II.1.1 Reconfigurable Architecture

For a given application set or domain, there are generally two implementation methods: ASICs and general purpose programmable/reconfigurable platform including Programmable Logic Device (PLD) and general purpose processor (GPP).

Because the functionality of the architecture is fixed, ASIC platform has exactly one-to-one correspondence between application and architecture. This fixed construction has no redundant parts, which makes the ASIC platform the most efficient in area, power and the least delay among all possible implementation platforms. But its Nonrecurring Engineering (NRE) cost is very high. The mask cost is over millions pounds for the 60-nm technology and the design cost runs into as high as tens of millions of pounds as the dimensions of chips approach nano-scales. Besides, time to discover the design failures and repair them would be long, making time-to-market of product uncertain, which is even a more important factor than other cost and may lead to product failure regardless of its high performance.

PLD is an electronic device containing reconfigurable digital circuits which can be programmed for targeted applications by users. GPP is a processor in which the programs stored in the integrated memory can be easily modified according to the requirements of applications and drive GPP to realize the desired function. PLD and GPP rise naturally to deal with today's multimedia and communication applications. These applications are becoming larger and larger, making the chip real estate more and more costly. They can be easily found in smart phone integrating multiple cell phone standards (e.g., Global System for Mobile communications (GSM), Wideband Code Division Multiple Access (WCDMA),

CDMA2000, Time Division-Synchronous Code Division Multiple Access (TD-SCDMA)), wireless Local Area Network (LAN), Bluetooth, MPEG-1 Audio Layer 3 (MP3), MPEG-2, MPEG-4, Digital Video Broadcasting - Handheld (DVB-H), digital camera/camcorder, graphics, games, etc. The PLD devices and programs stored in the memory of a GPP can be easily programmed according to the changes of applications. Under the semiconductor technology trend that the increase of memory density is outpacing that of transistor, storing multiple programs is more cost-effective than fabricating much larger chips.

Furthermore, even for a single application, there has been a tendency that complexity is growing rapidly. Historically, the growing complexity of applications has triggered the digitization revolution in the 90's of the last century and has changed most aspects of human life. Now the further growing complexity, mainly dynamics this time, will very likely introduce another "softwarization revolution" in the next decade. One piece of evidence is the prevalence of embedded processors. Another famous step is the Software-Defined Radio [3]. A typical embedded processor is Graphics Processing Units (GPU) which emerged in graphics and games application domain. It evolved from special ASIC blocks to domain-specific processors with their own C-like high-level programming languages and even larger than GPPs [4].

Compared with ASIC platforms, there are also other reasons for preferring a programmable platform. Firstly, a programmable solution greatly saves NRE cost [5]. It reduces not only mask cost but also the design cost since design efforts would shift from expensive hardware design to relatively cheap software design if the programmable platform is available. Secondly, programmable solution reduces uncertainty and risk. Software design takes less time than hardware design and its failure takes less time to discover and repair than hardware. This greatly reduces the time-to-market which is an even more important factor than cost.

II.1.2 FPGA devices

The general way to implement reconfigurable architecture is to adopt a PLD style core in the SoC design [6-9]. There are two basic architectures, namely, Complex PLD (CPLD) and FPGA. Using an embedded PLD is the mainstream method for a reconfigurable SoC.

Most FPGA devices are traditionally homogenous arrays of fine-grain, such as [10] and [11], which give the most possible flexibility. In fine-grain reconfigurable architectures the functionality of the hardware is specified at the bit-level or bits-level (less than four) and the programmable interconnection is manipulated as individual wire. The flexibility of fine-grain architecture comes at the cost of additional silicon area and this overhead hampers the performance of word-level algorithms like multiplications. Fine-grained architectures are efficient for bit-level masking and filtering or complex bit-oriented computations. Therefore word-level operations will become relatively large and slow when they are implemented with fine-grained architectures. To cover the gap between fine-grained FPGA devices and coarse-grained (word-level) reconfigurable architectures, 6-input Look-Up-Tables (LUT) based FPGA devices from Xilinx Virtex-5 family [12] are developed to meet the requirements of large complex applications for heavy load.

Compared with 4-input LUT fine-grained FPGA devices, Xilinx Virtex-5 FPGA devices show their merits in area and power efficiency for the computing-intensive digital signal processing applications. Such applications often require Random Access Memory (RAM)/ First In, First Out (FIFOs), mass of adder, subtracter, accumulator and multiplication, all of which are just integrated in Xilinx Virtex-5.

II.1.3 Domain-specific Processor

Compared with the poor configuration flexibility of ASIC platform, the flexibility of GPP platform will never be a problem. Actually the processor could implement any target application along with corresponding programs. But GPP platform really suffers from its abundant flexibility and thus is limited to several categories such as superscalar, Very-Long-Instruction-Word (VLIW), multithread, etc., which have been proved hard to maintain the annual performance increase of 50% [13].

More and more, power consumption and flexibility are becoming bottlenecks in VLSI design. Domain Specific Processor (DoSP) technology emerged to meet the simultaneous demands for flexibility and efficiency. The appropriate constraints imposed onto application sets can

release the architecture from the burden caused by the unnecessarily abundant flexibility, thus opening much larger design space for higher performance than GPP.

The lower flexibility in DoSP necessitates larger configuration bits (e.g., 8 contexts per plane for MorphoSys) compared with one 32-bit instruction for X86 and longer time (e.g. thousands to millions of cycles for FPGA) compared with one cycle for ARM in order to change the configuration. Of course, longer-lasting configuration set makes DoSP work more efficiently and reduce the cost of changing the configuration, which could be achieved by carefully optimizing the configuration set. This type of processor is a sort of reconfigurable architecture by definition. From the above, we observe that DoSP inherently leads to reconfigurable architectures either fine-grained (e.g., FPGA) or coarse-grained (e.g., MorphoSys), and vice versa.

GPP reconfigurable platforms can be roughly divided into fine grain architecture and coarse grain architecture in term of granularity which is determined by the width of the components in its datapath. Generally, an architecture is considered as fine-grain one when its datapath width is four bits or less [14]. Otherwise, it is considered coarse-grain architecture. In the following three sub-sections, fine-grain FPGA Architecture, typical fine-grain and coarse-grain architectures are discussed briefly.

II.2 FPGA Architecture

II.2.1 Fine-grain FPGA Architecture

The architecture model of FPGA is shown in Figure II-1. In this model, the reconfigurable hardware platform consists of three basic elements: Configurable Logic Blocks (CLBs), Connection Boxes (C-Box), and Switch Boxes (S-Box). The operational elements in CLBs of FPGA are mainly LUTs with 16 single bit inputs, which store the truth tables of user-defined combinational logic functions. These inputs are controlled by bits from the configuration memory which makes it possible to build any 4-input logic function by changing the content of the Static Random Access Memory (SRAM) configuration memory [15]. A combinational logic function is realized by looking up the value stored in the LUT that is addressed by the corresponding gate inputs. The programmable elements also have the

ability to optionally register their outputs. Furthermore, a mesh of programmable interconnects is available to connect the CLBs together to build bigger circuits.

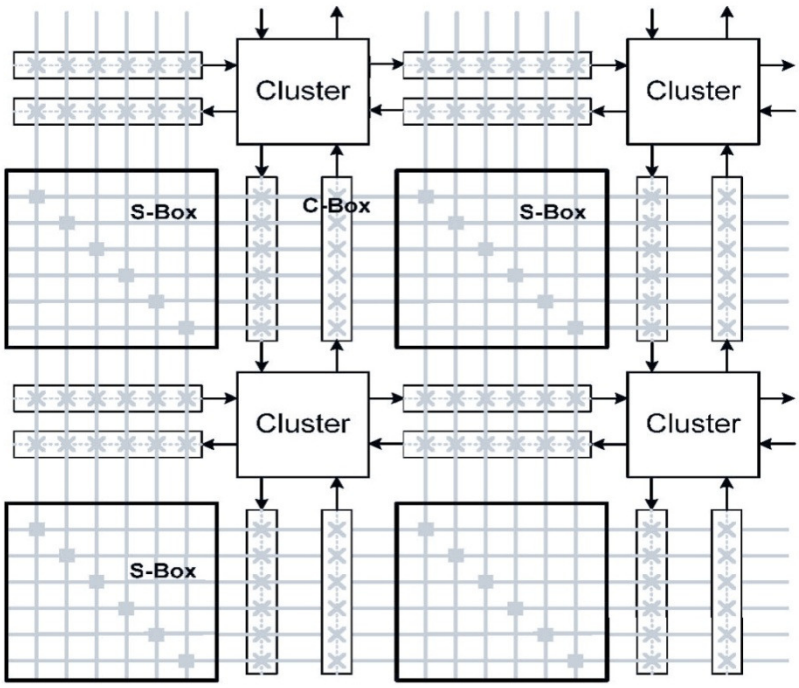

Figure II-1 : Architecture of FPGA

Implementing a logic network requires connecting CLBs by selecting the desired signal wire linked to the routing tracks through horizontal and vertical wiring channels located between two neighbouring rows or columns. A connection block can attach the signals to the logic block and the switch box nearby. The connections in the switch boxes make the input signal either pass through the switch box on its track or change its routing direction. To enhance the connectivity for connecting various CLBs, it is possible to use various types of wires with different lengths which are separated by variable numbers of blocks [14]. For example, in Xilinx's Virtex FPGAs, there are two types of routing devices, C-Box and S-Box, which route the signal flows among CLBs and wires. C-Box route the inputs and outputs of a CLB to the adjacent wires. S-Box connects horizontal and vertical crossing wires. The single-lines connect adjacent CLBs, while 16 lines connect CLBs that are three or six blocks apart [10].

The fine-grain aspect of FPGAs makes them extremely flexible and suitable for a very wide range of applications. Hence, FPGA chips are produced in large quantities which make their usage come with greatly reduced NRE costs. This high flexibility is obtained at the cost of very high power consumption which prohibits the deployment of FPGAs in portable applications.

II.2.2 Problems with Fine-grain FPGA Architecture

One problem with fine grain FPGAs is the high reconfiguration time. Take Atmel 40K40 as an example, which is a 48 by 48 FGPA and it needs 42063 8-bit words for full reconfiguration with maximum 8 MHz reconfiguration clock [16]. Therefore it can be fully reconfigured in 5.26 milliseconds. Similarly, Xilinx Virtex-E FPGA family has the array size of 64 by 96 with 766042 bits bitstream for full reconfiguration. It needs 3.1 milliseconds for reconfiguration by using a 50MHz clock [10]. The high reconfiguration time can be a restriction if dynamic reconfiguration is adopted in applications where parts of the circuit mapped on the FPGA are idle waiting for another part to finish. Dynamic reconfiguration of the circuit in this case would lead to better use of the available silicon. Besides, FPGAs usually have around 10 times more delays than ASICs.

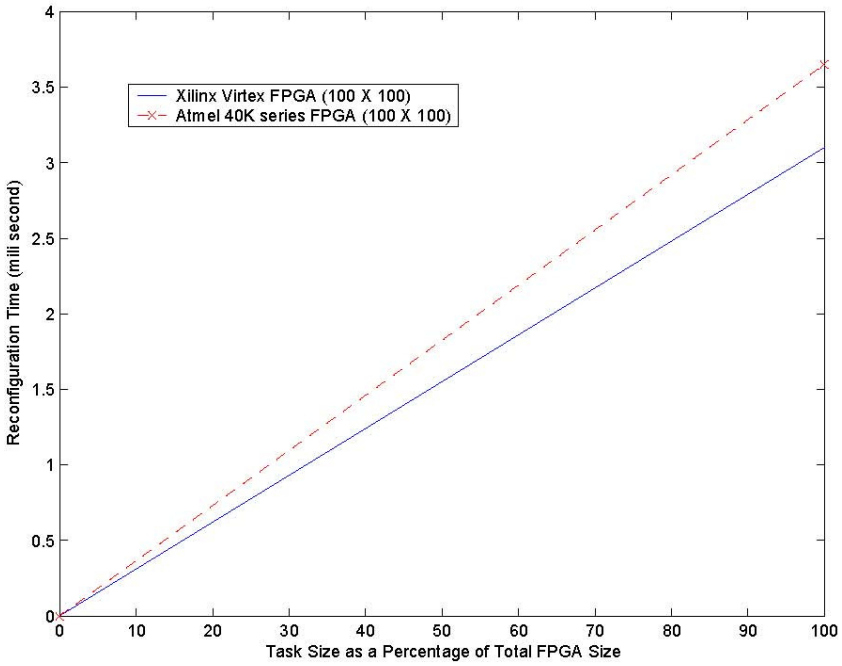

Figure II-2 : Reconfiguration time of two FPGA devices

Figure II-2 gives comparison of reconfiguration times of imaginary Atmel 40K and Virtex-E FPGA with 100 by 100 array size and reconfiguration clock of 50 MHz [17, 18].

In addition to the high reconfiguration overheads, FPGA devices also suffer from high power dissipation. In an FPGA chip, the energy dissipated in interconnects is about 65% of the total energy consumption, while 30% are dissipated in programmable clock-routings and I/O blocks. For example, the power consumption of an XC4085 chip running at a system clock of 50 MHz is approximately 6W [19]. Therefore, the high power consumption of FPGA is a limiting factor in energy-sensitive domains for the hand-held and portable devices such as wireless communication terminals, personal digital assistants (PDAs), laptops, etc. High power consumption directly shortens the running time of portable devices which, in a sense, directly determines the future of devices in the market.

As the size of the application becomes larger, the size of FPGAs has been growing steadily over the past decade and will stay on this path. This has been made possible by staying at the forefront in terms of the process technology. The combined effect of smaller feature sizes and larger die area is that more and more transistors are integrated on a die. The resulting increase in power density and total power dissipation will have an adverse effect even in the power insensitive domains, due to the advanced packaging and the cooling techniques required [19].

II.3 Fine-grain Architecture

Fine-grained architectures can make designers to take the benefit for implementing bit manipulation tasks flexible without wasting reconfigurable resources. The fine granularity of such architectures makes the implementation of large and complex calculations consuming numerous Processing Element (PEs). This results in slower clock rate when the applications are implemented with fewer, coarse-grained PEs.

To author's opinion, the number of fine-grain architectures is limited so it is difficult to classify them. Though each has its specific characters that differentiate them from others, they have a lot in common. Thus three popular architectures are randomly selected and introduced in this section. It is not intended to detail all industrial reconfigurable systems and

research projects; instead it is to show the overall characters of fine-grain architectures by introducing selected systems.

II.4.1 National Semiconductor's Adaptive Processing Architecture (NAPA)

NAPA [20-22] was developed by National Semiconductors USA. Adaptive Logic Processor (ALP) in the architecture couples a standard processor on-chip, 32-bit Reduced Instruction Set Computer (RISC) core (Compact RISC) called the Fixed Instruction Processor (FIP), with a reconfigurable array of fine-grained logic elements. Both ALP and FIP can access the same memory space and ALP, therefore, retains complete generality. The structure of NAPA processor is shown in Figure II-3. Additionally, the ALP has exclusive access to a set of configurable I/O pins and on-chip memory resources and a general external memory interface. This increased flexibility in interfacing and memory allocation are greatly helpful for adaptive computing effectively, especially in embedded systems.

To overcome the problems of consistency and synchronization between ALP and FIP, there are synchronization mechanisms such as standard status flags and interrupts with two programming modes for FIP and ALP threads to rejoin. In the first mode, FIP initiates the ALP operation and suspends afterwards. Once the work is done, the ALP reactivates FIP by an interrupt. In the second mode, the FIP is free to perform computation loaded after having initiated ALP.

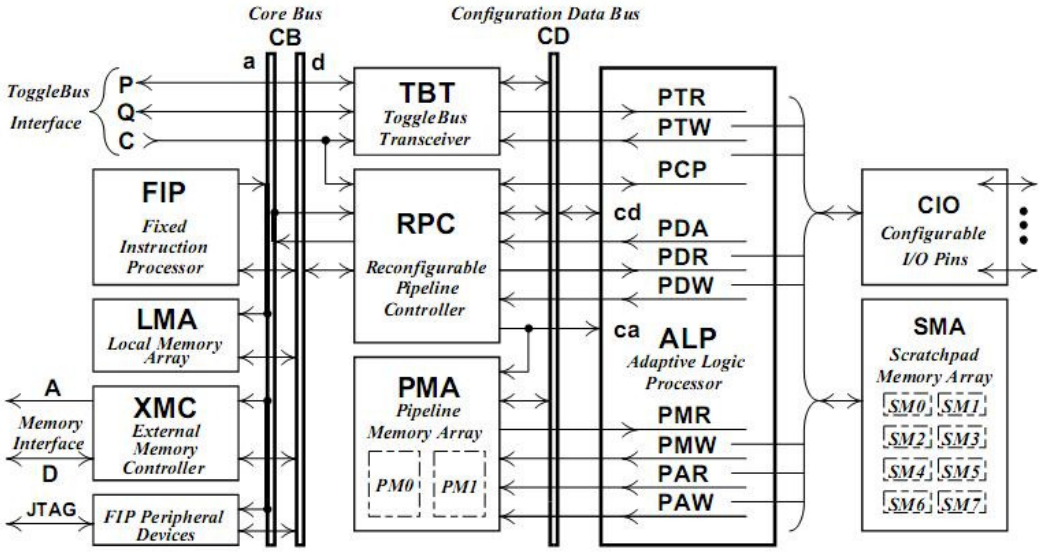

Figure II-3 : NAPA processor structure [20]

II.4.2 Garp: Gate Array Processor

Garp was developed by Berkeley Reconfigurable Architecture, System, and Software (BRASS) group at UC Berkeley, USA [23-25]. It is reserved for tasks such as interrupting the main processor, array-initiated memory accesses, and register transfers with the host processor. The Garp processor architecture, as shown in Figure II-4, combines a standard Microprocessor without Interlocked Pipeline Stages II (MIPS-II) processor with a two-dimensional reconfigurable array such as FPGA-like blocks available from Xilinx, Altera and other manufacturers. The reason to classify this processor into fine-grain architecture is 2-bit operands in size at most for each computing element in the reconfigurable array which is used to accelerate certain computations. Garp's main processor executes an extended MIPS-II instruction set and the reconfigurable array in it exchanges data between memory and the main processor through 4 memory buses which is vertical through the rows. In Data Encryption Standard (DES), image dithering, and a sorting algorithm, Garp processor runs at 133 MHz and speeds up from 2 to 24 over a 167 MHz UltraSPARC processor.

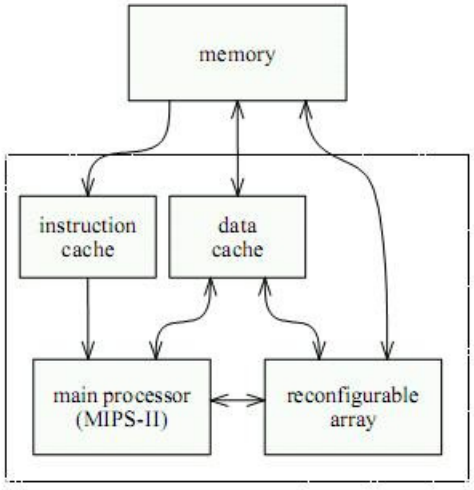

Figure II-4 : Garp block diagram [23]

The main problem is a lack of memory to store intermediate data inside the reconfigurable array. There are flip-flops only inside the logic elements, which can be used for memory resources. The intermediate data generated in reconfigurable array must be written back to the data cache. Due to the caching mechanism, accesses to intermediate data may cause

misses and stall array execution. Besides, large amount of data caching requires more bandwidth and therefore forms a bottleneck for its limited buses.

II.4.3 Chimaera Architecture

The Chimaera [6, 26, 27] architecture was developed by Scott Hauck and other researchers at Northwestern University first and University of Washington later. Figure II-5 demonstrates the overall Chimaera architecture. It consists of tightly coupled fine-grain reconfigurable functional units (FPGA blocks) and microprocessor for hardware caching. This integration was intended to eliminate the communications bottleneck between the two and allow the acceleration of a broad class of functions as opcodes. Functions loaded into the FPGA by the host processor operate speculatively on every clock cycle, with the results written back to a register file only for those functions explicitly invoked by the processor.

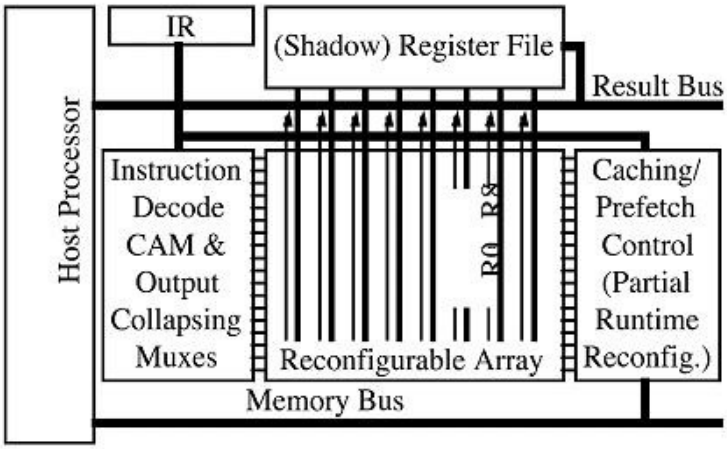

Figure II-5 : Overall Chimaera architecture [6]

The execution model is that of bit-slice data which spreads horizontally across the array, propagating downward as row-wise operations are applied. Kernels corresponding to operating codes take up entire contiguous rows. Hence, the array is partially reconfigurable by the row. Logic blocks have multiple LUTs and multiple inputs/outputs, allowing data forwarding while computing. Special carry-propagation logic propagates critical paths along each row. Horizontal wires of various reaches are available, but limited. In order to simplify context switching, there are no pipeline registers apart from the logic block's ability to

read/write the register file of the host processor. Besides, there are not pipelining latches in the reconfigurable array.

II.4 Coarse-grain Architecture

Coarse-grained reconfigurable architectures contain word-level function units, such as multipliers, Arithmetic Logic Units (ALUs) or PE which can perform a limited number of 16-bit or 32-bit operations as configured. One characteristic of coarse-grain architectures is the large size of PE and limited functions available in it. Compared with fine-grain architectures, coarse-grain architectures will consume less power, but it also suffers from the difficulty in implementing the control logic which is operated at bit level.

Generally, Coarse-grained architectures require less configuration data than fine-grained architectures because of their short configuration time. A survey of coarse grain architectures can be found in [28, 29]. The reconfiguration overhead of the coarse architectures is less than that of the fine grain architectures [30].

Coarse-grain architectures share certain features as introduced above while each has its own differentiating characters. Given their limited number and varied functions, it is difficult to systematically classify coarse-grain architectures. Thus five coarse-grain architectures are selected randomly as examples, which could make overall characters of coarse-grain architectures better revealed.

II.4.1 Pleiades Architecture

Pleiades [31] is an important vision for InfoPad project by Jan Rabaey's group at Wireless Research Centre at the University of California in Berkeley, which works toward multimedia on the mobile unit of computing services available from a high-bandwidth backbone network of computers, in which the driving need was to perform many different computationally intense tasks with low power.

The architecture, as shown in Figure II-6, consists of a general-purpose microprocessor for controlling and a surrounding heterogeneous array of coarse grain satellites. The main processor executes control-dominated sections of the program while satellites execute data-

dominated computations. The system is distributed in a sense that every satellite has its own instruction fetch and execution. The satellites communicate between each other through dedicated interconnects. The satellite processors could be arithmetic modules such as multipliers, memory modules, address generators or reconfigurable arrays [9, 32].

Because the configurable modules are function-specific, the paradigm is based on an ASIC flow. However, the control core processor is linked to satellite processors of varying degrees of specialization through a reconfigurable communication network. The task of architecture design and the decision of which satellites to use have to be done manually. Although dynamically configurable fine grain programmable gate array (PGA) satellites could accelerate functions, they cannot warrant special-purpose satellites and are less efficient than specialized circuits in the satellites. At partitioning stages the designer decides which loops of the full high-level program need to speed up using reconfigurable fabric; then the decision of which satellites to be deployed is made and their design started.[33]

Interconnects and the type/number of satellites can be parameterized to provide limited reconfigurability according to the requirements in applications. But programming the satellites requires writing low-level netlists. This technique can make the architecture efficient; however, they become too specific for diverse targeted applications.

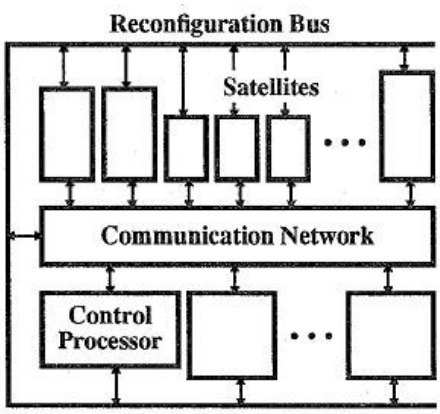

Figure II-6 : Overall Pleiades architecture [9]

II.4.2 RaPiD: Reconfigurable Pipelined Datapath

The RaPiD (Reconfigurable Pipelined Datapath) architecture [34], developed in 1996 at the University of Washington, USA, is a one-dimensional array of cells. It is aimed at speeding up highly repetitive, computationally-intensive tasks in multimedia and digital signal processing domain by implementing deep, application-specific computation pipelines to form a mostly linear pipeline in the RaPiD architecture.

An abstract view of the Rapid architecture is shown in Figure II-7. For RaPid-1prototype, each cell comprises an integer multiplier, three integer ALUs, six general purpose datapath registers and three local memories with 32 entries. Cells are connected by segmented buses with built-in pipeline registers and FIFOs at each end. The registers, the RAM, the ALU and all datapath operate on 16-bit data types. The multiplier performs a 16 x 16 to 32 multiplication and outputs the 32-bit result as two 16-bit words. The ALUs can be cascaded for double-precision operations. The data-path registers can be used to store constants or temporary values, to implement additional multiplexers, to support routing, and for additional pipeline delays. Each memory has a specialized datapath register featuring an incrementing feedback path. A RaPiD array is constructed by replicating identical cells from left to right, forming a linear computing pipeline. The array can consist of hundreds of cells, such as multipliers, adders, and comparators.[35-37]

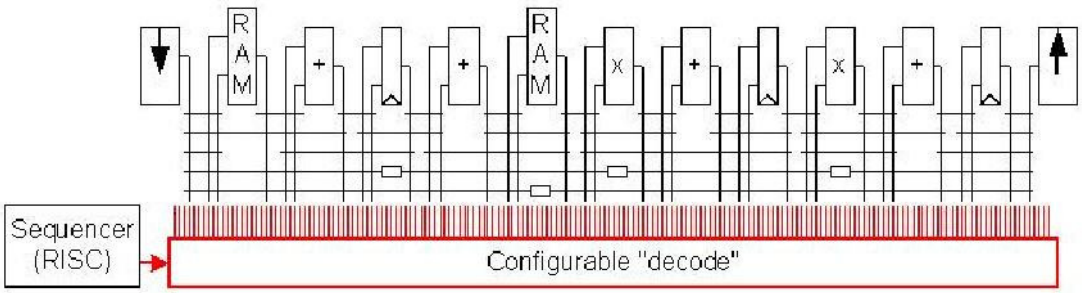

Figure II-7 : Abstract view of the Rapid architecture [34]

RaPiD is the only architecture that defines a broad architectural approach and provides a heterogeneous computation fabric/array. For configuration, the RaPiD architecture uses a

mix of static configuration and dynamic control, which brings plenty of flexibility for designers to configure it for their target domain.

Mapping applications onto RaPiD needs a non-standard programming language, RaPiD -B language, and compiler tool set, RaPiD -B compiler. All this causes the architecture to be considerably weak. Compared to an FPGA or MATRIX routing architecture, RaPiD restricts the connectivity among the processing elements to a linear segmented bus. This places the designer at a significant disadvantage when the architecture is configured for the target applications.

II.4.3 MorphoSys

Morphosys [38] is a reconfigurable processor that is a parallel system on one chip comprising a software programmable processing unit and a reconfigurable hardware unit from UC Irvine. It is targeted at the applications with inherent parallelism and a high level of granularity, which can be accelerated by the reconfigurable part. The granularity of the Morphosys processing elements is the highest among the family of reconfigurable processors. The performance of the Morphosys architecture on MPEG2, Motion Estimations, Discrete Cosine Transform (DCT) and Viterbi is around a 5-10 times improvement over normal Central Processing Units (CPUs) [39, 40].

The complete MorphoSys architecture, as shown in Figure II-8, consists of a MIPS-like TinyRISC core processor, a frame buffer, a Direct Memory Access (DMA) controller, a context memory, and an 8x8 reconfigurable array. The main component of Morphosys is the reconfigurable array of 8 by 8 which has a 28 bit, fixed point ALU (with a 16 x 12 multiplier) and a register file, and is configured through a 32-bit context word. The ALUs run on RISC instructions and the instruction set has load and store instructions for manipulation of the DMA controller and the reconfigurable array. The context word is loaded into a register in every execution cycle from the context memory, which is used to store the configuration data for the reconfigurable array including the functionality of the ALU (the instruction fetch and decode phase) and the network connections for that ALU. The frame buffer is used as data cache to store internal data for blocks of intermediate results.[41-43]

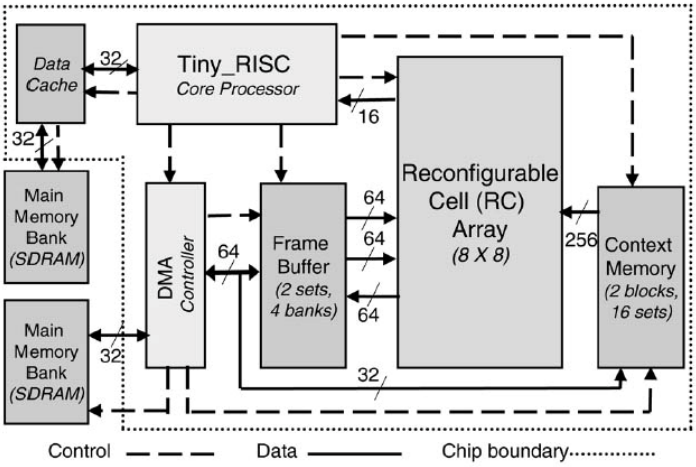

Figure II-8 : The MorphoSys Architecture [38]

The processor uses a hierarchical routing architecture, and is therefore capable of providing good routing flexibility. It follows that the Single Instruction, Multiple Data (SIMD) model and all the functional units in the same row or column execute the same operation with different data. Hence the array is only useful for data-parallel operations such as pixel parallel-data operations. In addition to its preference for word-level applications, which is caused by the coarse granularity of the processing elements, the architecture is also flexible enough to support bit level operations such as control operations which are executed by the RISC.

The Morphosys approach is well suited to some regular computation patterns but it does little to address the increasingly irregular patterns in the latest media standards, such as MPEG4. The regularity and simplicity of the reconfigurable array have limitations on implementation of some media processing algorithms and the applications with time-varying computation patterns. For example, an implementation of FIR on such an array is likely to cause excessive stalls and repeated-redundant context reloading. Besides, the architecture was not designed to be customised in spite of its synthesisable core.

II.4.4 Chameleon

The Chameleon reconfigurable processor [44], developed in University of Twente, Netherlands, is a heterogeneous reconfiguration architecture in combination with a Quality of Service (QoS) driven operating system.

It provides a platform for high-performance telecommunication and data communication applications. Chameleon reconfigurable processor is a general processor-based reconfigurable architecture, in which a 32-bit RISC core, the reconfigurable fabric, a fast bus, the local memory system, Programmable Logic Arrays (PLAs) for the control path and I/O are built in a single chip. [45]

The RISC core is employed as a host processor which schedules computation intensive tasks onto the programmable logic. The programmable logic is the main computing engine in the fabric, which is a 32-bit array of 108 data path units (DPUs). These DPUs are also capable of parallelling 16-bit operations and can be dynamically reconfigured between one and eight instruction execution. The main computational block within the DPU is an ALU capable of two's complement arithmetic and bitwise Boolean operations. It can simultaneously monitor and flag a number of relational and arithmetic conditions. The ALU is fed by two operand paths with optional Boolean masking, shifting, and registers for pipelining or storing constants. Input muxes select the operands from buses driven by other DPUs. Data can be read/written to/from the adjacent SRAM; each SRAM memory has one read port and one write port on the fabric side which can be used by the executing kernel. The SRAM memories can be chained together into a contiguous address space.[46-49]

A configuration bit stream is stored in the main memory and loaded onto the fabric at runtime by DMA. The programme logic can be configured at running time by bits stored in the memory. There are two kinds of planes in the DPUs: active and back planes. An active one executes the working bit stream and a back one contains the next configuration bit stream. It only takes one cycle to switch from the back plane to the active one. Therefore, the back plane can be treated as a cache for loading configurations.

II.4.5 RAP

The Reconfigurable Algorithm Processor (RAP) [50-52], originally developed by Bristol-based Elixent Ltd, is a coarse-grain reconfigurable platform designed for DSP and multimedia applications. The reconfigurable hardware, known as the D-fabrix, is made up of an array of hundreds of 4-bit ALU's and register/buffer blocks that can be cascaded together

to accommodate larger data lengths. This allows the fabric to operate on the 8-24 bit data lengths common in multimedia applications. The ALU's are arranged in a chessboard style, alternating with switchboxes which can act as a cross-point switch or 64 bits of configuration memory.

The D-Fabrix architecture is an extension of the Chess project [53] which is developed by Hewlett-Packard Laboratories. The logic in D-Fabrix ALU and Switchbox pair is shown in Figure II-9. Each basic 4-bit processing element of it has two 4-bit data inputs, one 4-bit data output, 1-bit carry input and carry output terminals to create carry chains linking between ALUs for wider words processing and a 4-bit instruction input. Besides, the array also contains 256-byte memory blocks dispersed around the array. The choice of nibble sized ALU's means that only a few bytes of memory are required to configure each ALU allowing rapid reconfiguration and improved density. The large amount of on-chip memory also allows ALUs to be fed by instruction streams generated within the array reducing off-chip memory traffic to improve overall performance.

Multiplexers are adopted in the switchbox in D-Fabrix, which is used to construct 2-input logic gates. One of the input ports of the multiplexer are connected to the control input and the other port is connected to data inputs. Although one of the four available data bits is used, the adoption of multiplexers achieves good performance in terms of area and speed compared with ALUs because of its efficient construction for simple Boolean logic implementation.

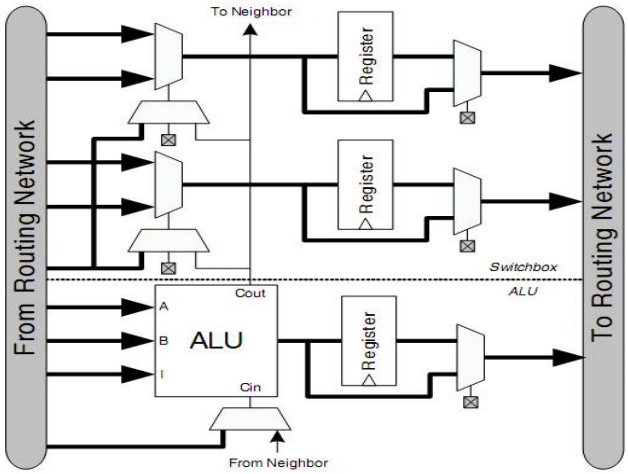

Figure II-9 : The logic in D-Fabrix ALU and Switchbox pair [52]

The RAP is targeted at multimedia and wireless base-station applications and they have shown a speed-up of 238x against a 32-bit DSP processor and 38x against an FPGA in Joint Photographic Experts Group (JPEG) compression application.

II.3 Interconnection Structure

Reconfigurable interconnection networks are the important underlying hardware infrastructures of reconfigurable system. They not only provide the whole system with powerful flexibility to meet the requirements from applications, but also affect the area, time, and power efficiency. Reconfigurable interconnection networks implement required connections among functional blocks or components through reconfiguring programmable switches and basically consist of programmable switches and wire segments or channels [54].

In this section, three typical interconnection structures are evaluated which are symmetrical interconnection network, hierarchical interconnection network and binary (Fat) interconnection tree.

II.4.1. Symmetrical Interconnection Network

FPGA devices employ non-distinctive logic blocks which are embedded in a mesh of routing sea consisting of switch boxes and connect boxes. Each function module in FPGA devices including Boolean and routing function modules are the same. The routing network is organized in symmetrical and balanced style.

Logic blocks in FPGA are separated by vertical and horizontal channels. There are prefabricated parallel wire segments running between each pair of adjacent logical blocks in both the vertical and horizontal channels. A switch block is located at each intersection of a vertical and horizontal channel. When an FPGA is used to implement a Boolean function, a partitioning algorithm is used to decompose the Boolean function into some smaller sub-function so that each of them can be implemented by a single logic block. Then a placement and routing algorithm is employed to select a logic block for each sub-function, and the wire segments and switches are chosen to connect the selected logic blocks.

During the whole processing, the selection of logic block, switch box and connect box is no distinction except the limitation caused by the requirement of delay.

II.4.2. Hierarchical Interconnection Network

While being effective for local connections, the symmetrical interconnection network has the disadvantage caused by the coarse granularity architecture that distant routing wires make communication slow and expensive since a large number of programmable switches have to be traversed. This leads to the concept of the hierarchical interconnection network, which continues to exploit locality while reducing the cost of the long connections.

Hierarchical interconnection network is focused on the interconnection between the coarse block elements such as Pleiades architecture (see sub-section II.4.1) and tries to overcome the routing problems caused by blocks with different area sizes. It is useful in applications where data locality is high and only a few signals need to be sent across the chip.

The interconnection network is composed of two types of connections: global interconnects and local segmented mesh structure. Global interconnects provide long distant connection between any two parts of the array. Furthermore, switching activities of the lines are transmitted for long distances. Local segmented mesh structures in local blocks improve overall global interconnects as shown in Figure II-10, but it is difficult for them to adapt to heterogeneous arrays, for a 2D regular grid has to be found. The disadvantage of hierarchical interconnection network is that their switching elements are less generic than symmetrical interconnection network. Besides, the complexity of two routing methods in their respective partitions always makes mapping distribution hard to achieve an optimal status as originally expected.

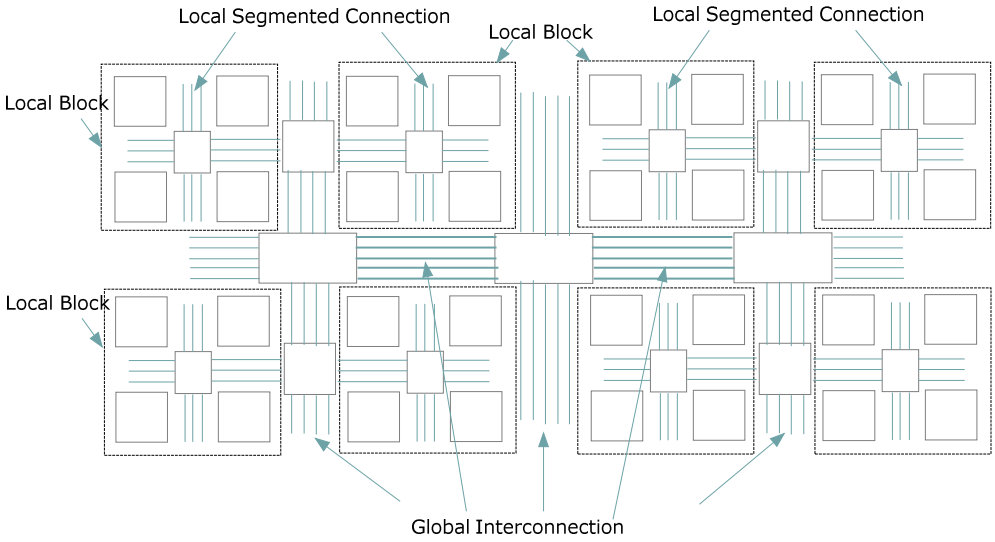

Figure II-10 : **An example of generalized hierarchical interconnection**

II.4.3. Binary (Fat) Interconnection Tree

The binary interconnect tree is a useful alternative to the hierarchical segmented mesh, which is also presented by the "fat tree". The hierarchical synchronous reconfigurable array used in the BRASS project [55] is such an interconnection which is a network based on a complete binary tree and the shared bus when logic block to logic block connections are needed. Binary trees have favourable features such as constant node degree, small node degree, scalability, etc. An example of such a network is shown in Figure II-11, in which, functional logic blocks, represented by the symbol PE, are located at the terminals of the network, like leaves on a tree. The routing nodes are represented by Rs. The number of buses per channel increases with the levels of the tree. For the communication between two local logic blocks, signals are transferred without the assistance of stem network bus which is in the upper level. The fat tree is more like a real tree, in which the branches get thicker toward the top.

The advantage of this architecture is that the number of switches used to route the signal grows logarithmically with the distance, which means that the overall delays introduced by the switches are lower.

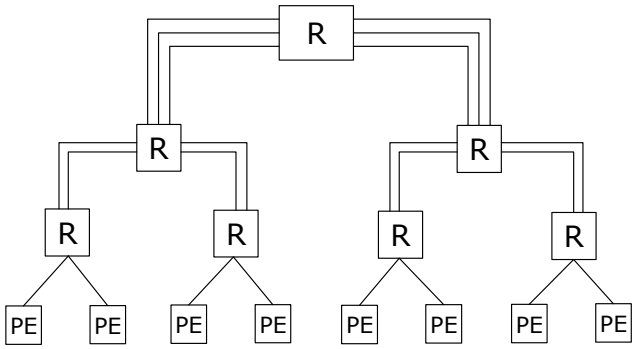

Figure II-11 : A fat binary tree

II.4 Low Power Technology

Traditionally, low-power VLSI design techniques have been focussed on improving energy efficiency. As the issue of low-power VLSI design techniques becomes more pervasive, the researches to minimize power consumption in VLSI design have been carried out on multiple lines: semiconductor technology, circuit design, system architecture, application design, design automation tools and operating system. Energy awareness is now gaining more and more attention in the mainstream VLSI design affecting all aspects of the design process.

Currently, most components in VLSI design are fabricated using CMOS technology. The reason for this bias is low cost and, inherently, efficient power consuming of Complementary metal–oxide–semiconductor (CMOS) technology compared with other technologies. There are two main components of power dissipation in a CMOS circuit: dynamic power and static power. During normal operation, the power dissipation of a CMOS circuit is determined by the dynamic power which is the switching energy consumption spent in charging and discharging of circuit nodes. The leakage currents of the circuit are orders of magnitude less than the operating currents when devices are switching and are negligible therefore. The dominant factor of energy consumption in CMOS is dynamic power consumption, which is linear with the effective switch capacitance, the square of supply voltage, and the frequency of operations.

Lower level energy consumption can thus be archived by reducing the supply voltage, the capacitive load and the switching frequency. Most low power technologies fall into these

three categories. The low power technology is explained as follows by introducing the mainstream low power design approaches.

- The most effective approach in minimizing power dissipation is to minimize the supply voltage, that is, to design systems which can run at as low a supply voltage as possible that will satisfy the performance requirements.
- An alternative approach could be adopted if the whole system cannot run at a low supply voltage, that is, to partition a system into certain independent circuits and to make each part run at its own optimal low supply voltage. It is an effective technique to minimize overall power while providing higher performance in processing elements that are timing-critical.
- The supply voltage can be dynamically adjusted to save the energy when the performance of a system is varied in some applications. This approach makes system always run at the lowest possible value that provides sufficient throughput.
- Concurrent processing technique can be applied to compensate the performance loss associated with lowering the supply voltage. Increasing the throughput of a given design by applying this technique could make the supply voltage be further decreased and power dissipation reduced while performance requirements still met.
- Minimizing capacitance is an important approach for reducing power dissipation. This technology is applied by adopting circuit blocks which are custom-made to perform specific computational task required by a given application. Versatile and general-purpose circuit blocks are generally less efficient for specific application.
- Avoid driving global signals across a chip and accessing large central memories and functional units. A targeted algorithm generally consists of a sequence of computational steps. Most computational steps typically interact and communicate with only a few previous and subsequent steps. The localized algorithms can be adopted to minimize the amount of power-hungry global interactions.
- Reducing switching activity is an effective technology to minimize power consumption since switching events are the source of energy consumption. This can be achieved by clock gating, minimizing glitching and powering down the sub-function modules when they are not actually demand. An effective coding scheme will also help a lot in some applications.
- Avoid hardware sharing which could destroy the temporal correlations present in path or data paths/streams. The frequent changing in interconnections and data buses will raise the power consumption dramatically.

To optimize a system for a certain application under an environment, a designer generally has to select a particular algorithm, design or use an architecture that can be used for it, and determine various parameters such as supply voltage and clock frequency. Therefore, the low-power design can be carried out in these aspects.

One approach to minimize power consumption is to adopt circuit blocks that are designed specifically to perform certain computational tasks required by a given application. This approach can significantly reduce the energy consumption for an operation because of the application-specific circuit blocks which are used to replace more versatile and general purpose circuit blocks. In a general purpose circuit block, redundant parts exit for certain application implementation. They are designed so that they can execute several different operations and necessarily larger and more complex than domain-specific circuit blocks. Besides, general purpose circuit blocks also have to be large enough to handle the largest data size for all possible given applications. The custom- designed circuit blocks remove these redundant parts to service the required operation only and consume less power than general purpose one.

II.5 Conclusion

This chapter has presented the basic concepts of reconfigurable architectures and the classifications of them. Fine-grain FPGA architectures and five typical coarse-grain reconfigurable architectures have been introduced in this chapter. Reference has been made to their features and the quality of design in which they may be carried out. Being an important part of reconfigurable systems, interconnection network and three typical implementations have been discussed. Lastly, the concepts about low power design technology are introduced at the end of this chapter.

Chapter III

Review of Distributed Arithmetic Algorithm and its Applications

III.1 Overview

Distributed Arithmetic (DA) has been widely adopted for its computational efficiency in many digital signal processing applications. The most frequently used form of computation in digital signal processing is a sum of products which is dot-product or inner-product generation. DA is generally a bit-serial computation operation that forms a product (dot or inner) of two vectors in one clock cycle. The typical applications include DCT, DFT (Discrete Fourier Transform), FIR (Finite Impulse Response), and DHT (Discrete Hartley Transform) which can be found in main stream multimedia standards and telecommunication protocols. All these applications involve inner products computation between two vectors, one of which is a constant.

The advantage of DA is its special non-multiplication mechanization which uses adder replacing multiplication and therefore simplifies the hardware implementation. This hardware characteristic limits its performances at the same time. The inherent bit-serial nature makes the DA apparent slower than the multiplication algorithms. The final result of DA will be obtained after N cycles where N is the bit width of input vectors. This disadvantage will not exist if the number of elements in the input vectors is commensurate with the number of bits in each vector element. For example, time required to input total eight 8-bit words parallelly which is one word each cycle is exactly the same as time required to input simultaneously all eight words serially [56].

The initial DA work dates back to 1968 by Zohar who had independently invented DA and applied it with FFT and digital filter [57-60]. DA is further developed with FIR and IIR digital filter mechanization by Peled, which is published in the IEEE ASSP Transaction [61, 62]. The most well-known description of DA was given by Abraham Peled and Bede Liu through a presentation on IIR digital filter mechanization at the Arden House Workshop on Digital Signal Processing in 1974. The complete definition of DA and full description of its application and implementation were described in the workshop. Numerous researchers made their contribution to DA applications and implementation after that. In 1989, Stanley A. White gave a detailed review of DA and its applications in digital signal processing [56].

This section will first review the definition of DA, followed by two basic DA extensions, ROM based DA and adder based DA. One of them, adder based DA, is the theory base of proposed processor in this dissertation. Three types of serial DA and one parallel DA architecture are discussed in the following sections including advantages and problems of constructions. Several typical DA applications will be briefly introduced at the end of this section.

III.2 Distributed Arithmetic Algorithm

III.2.1. DA Algorithms

DA is a bit-serial operation that computes the inner product of two vectors without using multiply operations. DA has an inherent bit-serial nature. Let us consider the computation of the following inner (dot) product with L-dimensional vectors:

$$Z = AX = \sum_{i=0}^{L-1} C_i X_i \qquad \textbf{(III-1)}$$

where A = [C_0, C_1,···, C_{L-1}] is an M bits fixed coefficient vector and X = [X_0, X_1,···, X_{L-1}] is an N bits input vector.

C_i and X_i are in two's complement binary scaled, then they can be expressed as follows:

$$C_i = -C_{i,(M-1)}2^{M-1} + \sum_{j=0}^{M-2} C_{i,j}2^j \qquad \textbf{(III-2)}$$

$$X_i = -X_{i,(N-1)}2^{N-1} + \sum_{k=0}^{N-2} X_{i,k}2^k \quad \textbf{(III-3)}$$

where $C_{i,j}$, $X_{i,k} \in \{0,1\}$ is the jth and kth bit of vector element C_i and X_i respectively. $C_{i,0}$, $X_{i,0}$ are the least significant bit (LSB) and $C_{i,(M-1)}$, $X_{i,(N-1)}$ are the sign bit. M is the word length of C_i and N is the word length of X_i.

To realize the inner product computation, the conventional DA uses a ROM-based architecture. Another method is to adopt an adder-based architecture.

III.2.2. ROM Based DA

ROM-based DA speeds up the multiplication process by pre-computing all possible values and storing them in a ROM.

By substituting Equation (III-3) in Equation (III-1), the output Z is given by:

$$Z = AX = \sum_{i=0}^{L-1} C_i(-X_{i,(N-1)}2^{N-1} + \sum_{k=0}^{N-2} X_{i,k}2^k)$$

$$= -\sum_{i=0}^{L-1} C_i X_{i,(N-1)}2^{N-1} + \sum_{k=0}^{N-2}\left[\sum_{i=0}^{L-1} C_i X_{i,k}\right]2^k \quad \textbf{(III-4)}$$

By defining the term R_k as

$$R_k = \sum_{i=0}^{L-1} C_i X_{i,k} \quad \textbf{(III-5)}$$

Then, we can obtain R_{n-1} when $k=N-1$

$$R_{N-1} = \sum_{i=0}^{L-1} C_i X_{i,N-1} \quad \textbf{(III-6)}$$

Equation (III-4) can be written as following format by substituting Equation (III-5) and Equation (III-6).

$$Z = -R_{N-1}2^{N-1} + \sum_{k=0}^{N-2} R_k 2^k = \sum_{k=0}^{N-1} S_k R_k 2^k \quad \textbf{(III-7)}$$

where S_k is defined as the sign of term for k= 0,1,· · ·,*N-2*, *N-1*.

$$S_k = \begin{cases} -1 & k = N-1 \\ 1 & 0 \le k \le N-2 \end{cases} \quad \textbf{(III-8)}$$

Since $X_{i,k} \in \{0,1\}$, R_k has 2^L possible values for k= 0,1,· · ·,*L-1*. Rather than computing these values on line, these values can be precomputed and stored in a ROM. Then, Equation (III-5) can be implemented with a ROM of size 2^L.

The bits of input data ($\{X_{0,k}, X_{1,k},\cdots,X_{i,k}\}$) are used to form ROM addresses. The ROM contents following with an adder and register can realize the accumulation for k rising from 0 to *N-1*as shown in Equation (III-6). An arithmetic shifter in the accumulator feedback path is used to form successive scaling with powers of two. Then, after N cycles, corresponding to the bit-width of input vector *X*, the final value of output *Z* can be obtained as the result of the accumulation.

The serial processing pattern of ROM based DA becomes a bottleneck when the outputs data are expected for each clock cycle. For some real-time video-streams applications, the high data throughput is the critical feature. Another problem with ROM-based DA is that its ROM size (2^L word) grows exponentially as the order L increases. As the number of inputs and the internal precision become larger, the ROM-based DA will suffer from extremely large ROM requirements.

III.2.3. Adder-Based DA

From the arithmetic point of view, the adder-based DA is not much different from the ROM-based DA. In ROM-based DA, Equation (III-4) is obtained by substituting Equation (III-3) in Equation (III-1). Where, X_i, one of the two factors in Equation (III-1), is changed to two's complement binary format, the other factor, C_i, is led to keep its original format. For adder-based DA, the roles of the two factors are exchanged: X_i keeping its original format and C_i being changed to two's complement binary format. This role exchanging makes the DA processing pattern switch from serial to parallel.

By substituting Equation (III-2) in Equation (III-1), the output *Z* is given by:

$$Z = AX = \sum_{i=0}^{L-1} X_i(-C_{i,(M-1)}2^{M-1} + \sum_{j=0}^{M-2} C_{i,j}2^j)$$

$$= -\sum_{i=0}^{L-1} X_i C_{i,(M-1)}2^{M-1} + \sum_{j=0}^{M-2}\left[\sum_{i=0}^{L-1} X_i C_{i,j}\right]2^j \quad \textbf{(III-9)}$$

We define term T_j as

$$T_j = \sum_{i=0}^{L-1} X_i C_{i,\mathrm{j}} \quad \textbf{(III-10)}$$

then, Equation (III-9) can be written as

$$Z = -T_{\mathrm{M-1}}2^{M-1} + \sum_{j=0}^{M-2} T_j 2^j \quad \textbf{(III-11)}$$

$$= \sum_{j=0}^{M-1} S_j T_j 2^j$$

In theory, T_j can also be implemented with a ROM since $C_{i,j}$ is either 0 or 1, like R_k in ROM-based DA. But, the size of ROM will be far larger than the one in ROM-based DA. Since $C_{i,j}$ is fixed and known, T_j can be realized with adders. Clearly, only the inputs corresponding to nonzero coefficient bits $C_{i,j}$ need to be added. In theory, this characteristic will lead the number of adder to decrease by half on average.

III.3 Distributed Arithmetic Implements

Generally, DA applications fall into DSP algorithms which are implemented using general purpose programmable DSP chips for low-rate applications, or special-purpose DSP chip-sets and ASICs which are designed specifically for fixed function for higher rates.

The FPGA platform is another alternative method for implementation, which maintains the advantages of custom functionality like an ASIC while avoiding the high development costs and the inability to make design modifications after production. The FPGA also increases design flexibility and adaptability with optimal device utilization while conserving both broad space and system power, which is often not the case with DSP chips. When a design demands the use of a DSP, or time-to-market is critical, or design adaptability is crucial, then the FPGA may offer a better solution.

In this section, the discussion focuses only on the general architectures for DA implementation which are not restricted with certain platform such as DSP, ASIC, FPGA and so on.

III.3.1 Architecture for Distributed Arithmetic

A block diagram of a general architecture for DA is shown in Figure III-1, which is capable of 4 input vectors with 5-bit width. The architecture consists of a $2x2^L$ (L=4) ROM with a 5x32 decoder, register and a left shift-accumulator. $X_{i,0}$ are the least significant bit (LSB) of input vectors X_i (i=[0,3]); $X_{i,4}$ are the most significant bit (MSB) of input vectors X_i (i=[0,3]); S_k is defined as the sign of term as defined in Equation (III-8), whose value is -1 when k equals 4 and 0 when k equals other values. The memory must contain all possible 16 (2^4) values and their negatives in order to accommodate the value of S_k which occurs at the sign-bit time. As a result, a ROM with $2x2^L$ word size is required.

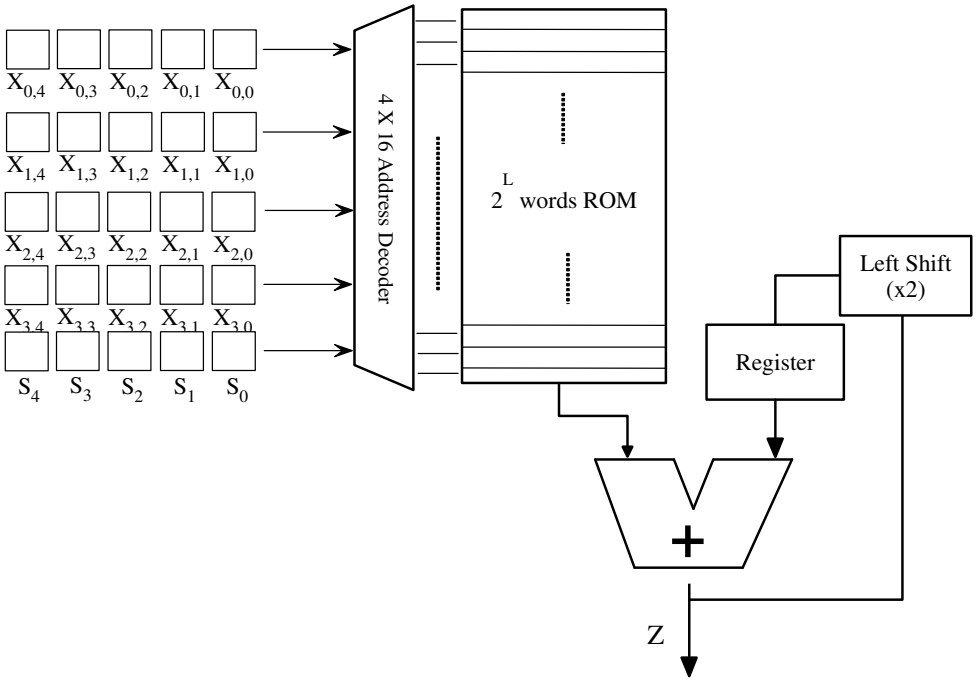

Figure III-1 : General architecture for DA

It can be seen from the Figure III-1 that the input vectors X_0, X_1, X_2 , X_3 and S_k are serial, 2's-complement numbers. With bit-serial input data at the LSB of word first, one bit of each L input of length N is used to address the ROM. The sign bit S_4 is the last bit to arrive. The clock period from the LSB reaching input ports to sign bits all simultaneously arriving is the

processing time. Within each clock cycle in the processing time, the input vectors together with S_k arrive at address decoder and the corresponding value stored in ROM is output to the adder followed. An adder, 1-bit left-shift unit and register construct a left-shift accumulator. The value from ROM is left-shifted one position (i.e. multiplied by two) and added with its next clock value. This is repeated until the sign bits are fed. Therefore, after a whole processing time, the fully formed result is output.

III.3.2 Memory Reduced DA Architecture

The ROM size in the original DA architecture can be reduced by half to 2^L word by replacing the adder to an adder/substractor.

Let us take an example to show how the size of ROM is reduced. It is supposed that $C_0 = 0.32$, $C_1 = -0.48$, $C_2 = 0.89$ and $C_3 = 0.65$. The memory in the original architecture must contain all 16 (2^L, L=4) possible values and their negatives according to all the input cases. All possible input cases, corresponding terms of coefficient vector C_i and values are listed in Table III-1.

It can be seen from the Table III-1, the absolute values in the upper half are the same as the ones in the lower half. Therefore, S_k can be used as the control signal for an adder/subtractor and only data in upper half of the Table III-1 is enough to implement DA computation. The memory-reduced DA architecture is shown in Figure III-2. This configuration is now mechanized with a 16-word ROM. The values stored in the ROM are simply the upper half of the original one.

Table III-1: The content in the ROM of original DA architecture

S_k	X_3	X_2	X_1	X_0	Sum of terms	Values (E_1)
0	0	0	0	0	0	0
0	0	0	0	1	C_0	0.32
0	0	0	1	0	C_1	-0.48
0	0	0	1	1	$C_1 + C_0$	-0.16
0	0	1	0	0	C_2	0.89
0	0	1	0	1	$C_2 + C_0$	1.21
0	0	1	1	0	$C_2 + C_1$	0.41
0	0	1	1	1	$C_2 + C_1 + C_0$	0.73
0	1	0	0	0	C_3	0.65
0	1	0	0	1	$C_3 + C_0$	0.97
0	1	0	1	0	$C_3 + C_1$	0.17
0	1	0	1	1	$C_3 + C_1 + C_0$	0.49
0	1	1	0	0	$C_3 + C_2$	1.54
0	1	1	0	1	$C_3 + C_2 + C_0$	1.86
0	1	1	1	0	$C_3 + C_2 + C_1$	1.06
0	1	1	1	1	$C_3 + C_2 + C_1 + C_0$	1.38
1	0	0	0	0	0	0
1	0	0	0	1	$-C_0$	-0.32
1	0	0	1	0	$-C_1$	0.48
1	0	0	1	1	$-(C_1 + C_0)$	0.16
1	0	1	0	0	$-C_2$	-0.89
1	0	1	0	1	$-(C_2 + C_0)$	-1.21
1	0	1	1	0	$-(C_2 + C_1)$	-0.41
1	0	1	1	1	$-(C_2 + C_1 + C_0)$	-0.73
1	1	0	0	0	$-C_3$	-0.65
1	1	0	0	1	$-(C_3 + C_0)$	-0.97
1	1	0	1	0	$-(C_3 + C_1)$	-0.17
1	1	0	1	1	$-(C_3 + C_1 + C_0)$	-0.49
1	1	1	0	0	$-(C_3 + C_2)$	-1.54
1	1	1	0	1	$-(C_3 + C_2 + C_0)$	-1.86
1	1	1	1	0	$-(C_3 + C_2 + C_1)$	-1.06
1	1	1	1	1	$-(C_3 + C_2 + C_1 + C_0)$	-1.38

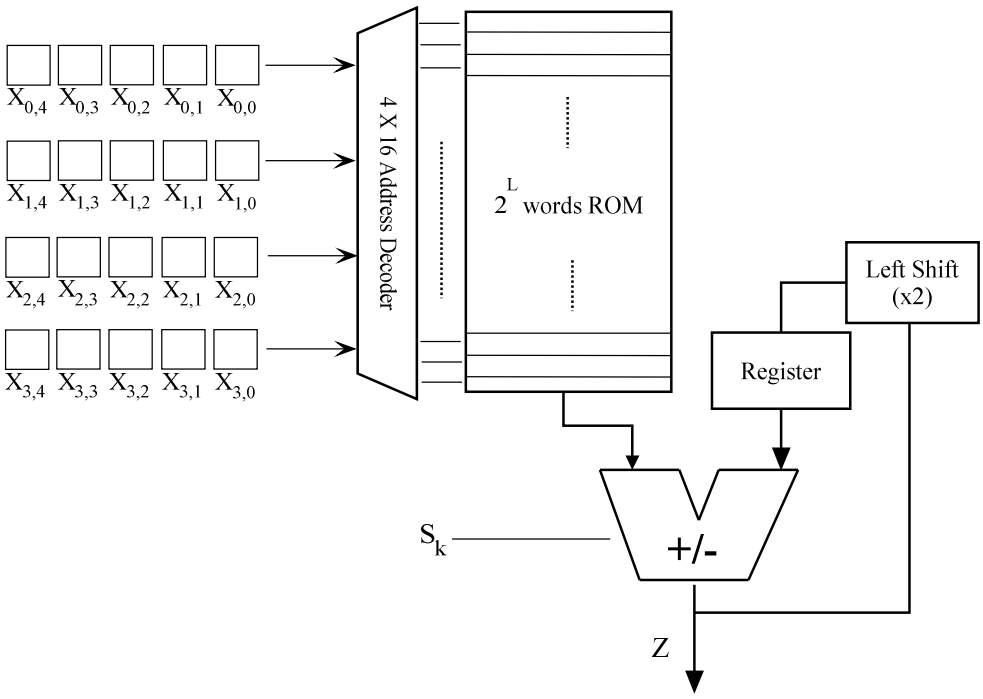

Figure III-2 : Memory reduced DA architecture I

From the hardware point of view, the architecture in Figure III-2 is quite efficient. The extra cost resulting from reduction of 16-word ROM is an additional subtractor. This benefit for hardware will be enlarged by saving 32 words on ROM for an 8-input vectors application.

The memory size may be further halved again to 2^{L-1} words. We currently call this implementation as memory reduced DA architecture II. It is supposed that there is a set of values E_2 which has one-to-one correspondence with input vectors $X = [X_0, X_1, X_2, X_3]$ as shown in Table III-2.

Table III-2 : The content in the memory reduced DA architecture II

X_3	X_2	X_1	X_0	Sum of terms	Values (E_2)
0	0	0	0	0	-0.69
0	0	0	1	C_0	-0.37
0	0	1	0	C_1	-1.17
0	0	1	1	$C_1 + C_0$	-0.85
0	1	0	0	C_2	0.2
0	1	0	1	$C_2 + C_0$	0.52
0	1	1	0	$C_2 + C_1$	-0.28
0	1	1	1	$C_2 + C_1 + C_0$	0.04
1	0	0	0	C_3	-0.04
1	0	0	1	$C_3 + C_0$	0.28
1	0	1	0	$C_3 + C_1$	-0.52
1	0	1	1	$C_3 + C_1 + C_0$	-0.2
1	1	0	0	$C_3 + C_2$	0.85
1	1	0	1	$C_3 + C_2 + C_0$	1.17
1	1	1	0	$C_3 + C_2 + C_1$	0.37
1	1	1	1	$C_3 + C_2 + C_1 + C_0$	0.69

It can be seen from the Table III-2, the absolute values in the upper half are the same as ones in the lower half. But the one-to-one relationship is different from the one in Table III-1. The relationship between upper and lower half values and their corresponding input vectors are shown in Equation (III-12).

$$E_2(X_2, X_1, X_0)\big|_{X_3=0} = -E_2(\overline{X}_2, \overline{X}_1, \overline{X}_0)\big|_{X_3=1} \quad \textbf{(III-12)}$$

The values E_1 in Table III-1 can be transformed to the values E_2 inTable III-2 as shown in Table III-3.

Table III-3 : Transform between Table III-1 and Table III-2

X_3	X_2	X_1	X_0	Values (E_1)	Transform	Values (E_2)
0	0	0	0	0		-0.69
0	0	0	1	0.32		-0.37
0	0	1	0	-0.48		-1.17
0	0	1	1	-0.16		-0.85
0	1	0	0	0.89		0.2
0	1	0	1	1.21		0.52
0	1	0	1	0.41		-0.28
0	1	1	1	0.73	− 0.69 =	0.04
1	0	0	0	0.65		-0.04
1	0	0	1	0.97		0.28
1	0	1	0	0.17		-0.52
1	0	1	1	0.49		-0.2
1	1	0	0	1.54		0.85
1	1	0	1	1.86		1.17
1	1	1	0	1.06		0.37
1	1	1	1	1.38		0.69

Where 0.69 is represented as Q_1 which is generated by:

$$Q_1 = \frac{(C_3 + C_2 + C_1 + C_0)}{2} = \frac{(0.65 + 0.89 - 0.48 + 0.32)}{2} = 0.69 \quad \textbf{(III-13)}$$

To implement this memory reduced DA architecture II as shown inTable III-2， Table III-3, Equation(III-12) and Equation (III-13), an initial register is required to store the value of Q_1 and several extra inverts are needed to realize memory reading according to the input vectors in the upper and lower half of Table III-2. Besides, an adder/subtractor is necessary which works in the way similar to the one in Figure III-2. Compared with memory reduced DA architecture I, the memory size of the architecture is reduced by half.

Following the same method, the memory size in Table III-2 can be further reduced in halves to 2^{L-2} words. We currently call the hardware implementation of it as memory of reduced DA architecture III. It is supposed that there is a set of values E_3 which has one-to-one correspondence with input vectors $X = [X_0, X_1, X_2]$ as shown in

Table III-4 : The content in the ROM of reduced DA architecture III

X_2	X_1	X_0	Sum of terms	Values (E_3)
0	0	0	0	-0.365
0	0	1	C_0	-0.045
0	1	0	C_1	-0.845
0	1	1	$C_1 + C_0$	-0.525
1	0	0	C_2	0.525
1	0	1	$C_2 + C_0$	0.845
1	1	0	$C_2 + C_1$	0.045
1	1	1	$C_2 + C_1 + C_0$	0.365

It can be seen from Table III-4, the absolute values in the upper half are the same as the ones in the lower half. The one-to-one relationship is similar to the one in Table III-2, which can be described as shown in Equation (III-14)

$$E_3(X_1, X_0)\big|_{X_2=0} = -E_3(\overline{X}_1, \overline{X}_0)\big|_{X_2=1} \quad \textbf{(III-14)}$$

The transformation between the values E_3 in Table III-4 and values E_2 in Table III-2 closely resembles that between Table III-1and Table III-2, which is shown in Table III-3.

Table III-5 : Transform between Table III-2 and Table III-4

X_2	X_1	X_0	Values (E_2)	Transform	Values (E_3)
0	0	0	-0.69	- (-0.325) =	-0.365
0	0	1	-0.37		-0.045
0	1	0	-1.17		-0.845
0	1	1	-0.85		-0.525
1	0	0	0.2		0.525
1	0	1	0.52		0.845
1	1	0	-0.28		0.045
1	1	1	0.04		0.365

Where −0.325 is represented as Q_2 which is generated by:

$$Q_2 = \frac{-C_3}{2} = \frac{-0.65}{2} = -0.325 \quad \textbf{(III-15)}$$

To implement this memory reduced DA architecture III as shown in Table III-4, Table III-5, Equation (III-14) and Equation (III-15), compared with memory reduced DA architecture I, an extra initial register is required to store the value of Q_2 and additional control parts are needed which act based on the value of X_3. According to these extra hardware costs, the

memory size is halved to 2^{L-2} words which is only one-fourth of the ROM in memory reduced DA architecture I.

III.3.3 Offset Binary Coding Architecture

In memory reduced DA architecture II and III, the memory reduction is obtained through adding or subtracting an initial value from original values. The initial value can be treated as an offset value. This recoding of the coefficient is denoted by Stewart G. Smith in [63] as Offset Binary Coding (OBC).

This method is based on a modified two's-complement representation of the values and reduces the memory size by a factor of two. The OBC can be further extended, reducing the memory size in steps by factor of two from 2^L to L in theory. However, this requires additional hardware in terms of adders and multiplexers, thus increasing the latency.

OBC uses a (-1, 1) offset binary code to replace a (0, 1) straight binary code as the format of input vectors. The input vector X_i can be expressed using an equivalent expression, as follows:

$$X_i = \frac{1}{2}[X_i - (-X_i)] \tag{III-16}$$

Being in 2's-complement notation, the negative of X_i is expressed as

$$-X_i = -\overline{X}_{i,(N-1)}2^{N-1} + \sum_{k=0}^{N-2}\overline{X}_{i,k}2^k + 1 \tag{III-17}$$

By substituting Equation (III-17) in Equation (III-16), Equation (III-16) can be re-written as

$$\begin{aligned} X_i &= \frac{1}{2}[-X_{i,(N-1)}2^{N-1} + \sum_{k=0}^{N-2}X_{i,k}2^k - (-\overline{X}_{i,(N-1)}2^{N-1} + \sum_{k=0}^{N-2}\overline{X}_{i,k}2^k + 1)] \\ &= \frac{1}{2}[-(X_{i,(N-1)} - \overline{X}_{i,(N-1)})2^{N-1} + \sum_{k=0}^{N-2}(X_{i,k} - \overline{X}_{i,k})2^k - 1] \end{aligned} \tag{III-18}$$

We define the term M_i as

$$M_{i.k} = X_{i,k} - \overline{X}_{i,k} \quad (k = 0,1,.....,N-2) \tag{III-19}$$

$$M_{i.N-2} = -(X_{i,N-2} - \overline{X}_{i,N-2}) \tag{III-20}$$

where the possible value of the M_i is 1 or -1. Then, Equation (III-18) can be simplified as

$$X_i = \frac{1}{2}(\sum_{k=0}^{N-1} M_{i.k} 2^k - 1) \quad \textbf{(III-21)}$$

By substituting Equation (III-21) in Equation (III-1), Equation (III-1) can be written as

$$\begin{aligned} Z = AX &= \sum_{i=0}^{L-1} C_i X_i \\ &= \sum_{i=0}^{L-1} C_i \frac{1}{2}(\sum_{k=0}^{N-1} M_{i.k} 2^k - 1) \\ &= \sum_{k=0}^{N-1} (\frac{1}{2} \sum_{i=0}^{L-1} C_i M_{i,k}) 2^k - (\frac{1}{2} \sum_{i=0}^{L-1} C_i) \end{aligned} \quad \textbf{(III-22)}$$

In order to simplify notation later, we define a function O_k and O_0 as

$$O_k = \frac{1}{2} \sum_{i=0}^{L-1} C_i M_{i,k} \quad \textbf{(III-23)}$$

$$O_0 = \frac{1}{2} \sum_{i=0}^{L-1} C_i \quad \textbf{(III-24)}$$

Then, Equation can be written as

$$Z = \sum_{k=0}^{L-1} O_k \bullet 2^k - O_0 \quad \textbf{(III-25)}$$

It can be seen from the Equation (III-25) that Z still can take 2^L values but only 2^{L-1} different magnitude values with a sign for O_k are consistent with the statements in the memory reduced DA architecture II. We use a 4-coefficient ($C_0 = 0.32$, $C_1 = -0.48$, $C_2 = 0.89$ and $C_3 = 0.65$) case as an example to show the results of Equation (III-25) in the format of Ci and the true values which are listed in Table III-6.

Table III-6: Expansion of Equation (III-25) for the case

X_3	X_2	X_1	X_0	Contents in format of C_i	True values
0	0	0	0	$-(C_3 + C_2 + C_1 + C_0)/2$	-0.69
0	0	0	1	$-(C_3 + C_2 + C_1 - C_0)/2$	-0.37
0	0	1	0	$-(C_3 + C_2 - C_1 + C_0)/2$	-1.17
0	0	1	1	$-(C_3 + C_2 - C_1 - C_0)/2$	-0.85
0	1	0	0	$-(C_3 - C_2 + C_1 + C_0)/2$	0.2
0	1	0	1	$-(C_3 - C_2 + C_1 - C_0)/2$	0.52
0	1	1	0	$-(C_3 - C_2 - C_1 + C_0)/2$	-0.28
0	1	1	1	$-(C_3 - C_2 - C_1 - C_0)/2$	0.04
1	0	0	0	$-(-C_3 + C_2 + C_1 + C_0)/2$	-0.04
1	0	0	1	$-(-C_3 + C_2 + C_1 - C_0)/2$	0.28
1	0	1	0	$-(-C_3 + C_2 - C_1 + C_0)/2$	-0.52
1	0	1	1	$-(-C_3 + C_2 - C_1 - C_0)/2$	-0.2
1	1	0	0	$-(-C_3 - C_2 + C_1 + C_0)/2$	0.85
1	1	0	1	$-(-C_3 - C_2 + C_1 - C_0)/2$	1.17
1	1	1	0	$-(-C_3 - C_2 - C_1 + C_0)/2$	0.37
1	1	1	1	$-(-C_3 - C_2 - C_1 - C_0)/2$	0.69

It can be seen that the true values in Table III-6 are exactly the same as the ones in Table III-2 and the values in the lower half are the mirror image of the values in the upper half. Using the Ci values of the example, the true value of O_0 can be obtained, which is

$$O_0 = \frac{1}{2}\sum_{i=0}^{4-1} C_i = \frac{1}{2}\sum_{i=0}^{3} C_i = \frac{1}{2}(C_3 + C_2 + C_1 + C_0) = 0.69 \qquad \textbf{(III-26)}$$

This value is the same as Q_1 defined in Equation (III-13). The coding format in Equation (III-25) can be seen as an offset value subtracting from initial values, hence being named as offset binary coding.

The architecture for OBC DA is shown in Figure III-3 which consists of 2^{L-1} word memory, a one-word register for offset value, an adder and a single adder/subtractor with the necessary logic gates for control.

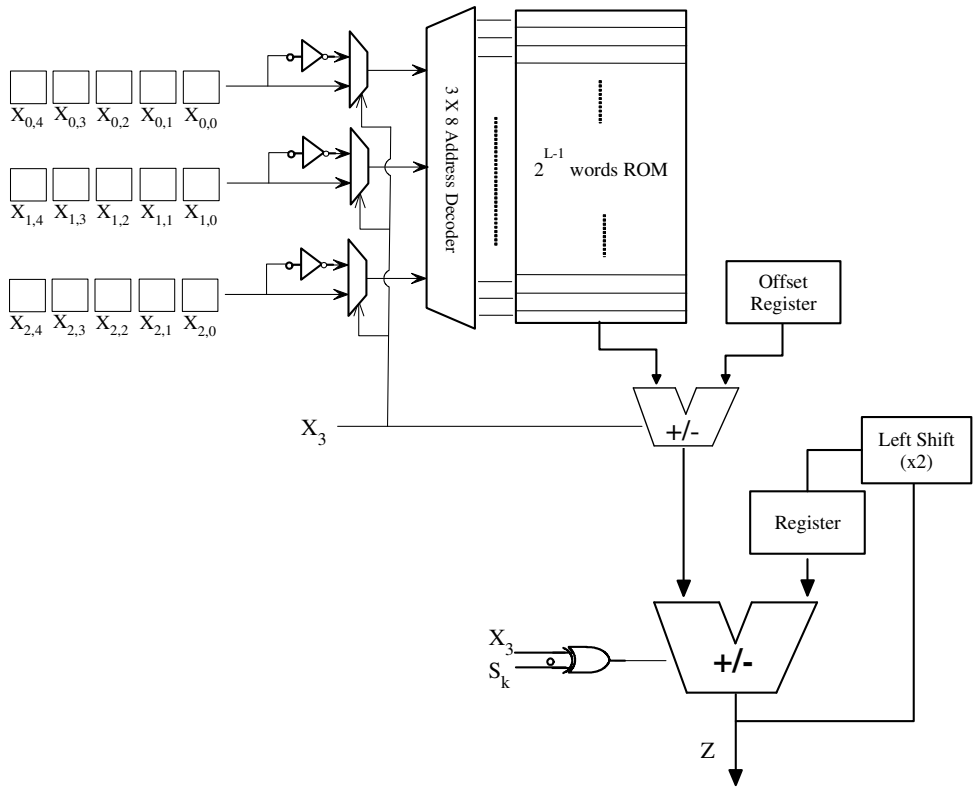

Figure III-3 : Memory reduced DA architecture II

The architecture shown in Figure III-3 is a schematic diagram. The control units between X_3 and X_i (i=0, 1, 2) can be synthesized as an EXOR gate. The offset register in the architecture stores true value of offset which can be pre-computed based on the coefficients of application.

Let us have a look at how the architecture works. The values stored in ROM are listed in Table III-6. The value in offset register is 0.69. It is supposed that the input vectors [X_0, X_1, X_2, X_3, S_k] is (11111). The input [X_0, X_1, X_2] and its complement value (000) are sent to the three 2-to-1 multiplexers. The input vector X_3 is used as the control signal for multiplexers and the complement values (000) of the inputs are output to address decoder. Now, we have the proper address for the ROM and the value -0.69 is correctly pulled out and sent to adder/subtractor which is driven by X_3 and configured as subtractor for this case. Then, the result of adder/subtractor is that value in offset register is subtracted from input. The value, -1.38, is obtained at the end of first adder/subtractor in the architecture. Until now, we have properly addressed the 8-word ROM, taken out the value which is further processed by

adder/subtract and gotten the final one. This value is exactly the same as the one in Table III-1 addressed as (11111).

This value will be passed to the second adder/subtractor which is driven by X_3 and S_k. After 1-bit left-shift and adder (subtractor), the final value of one clock cycle is stored in the register and will be involved in the operation with the value from next clock cycle. When N clock cycles' accumulation is done, the architecture will output the final result for DA computation.

III.3.4 Parallel DA Architecture

The approaches described above correspond to a bit Serial Distributed Arithmetic (SDA). All possible linear combinations of the constant coefficient elements C_i are stored in a ROM. The input vectors X_i is used to form the ROM address with LSB first. No matter how much ROM is required, the final result is available in a bit parallel format after N cycle where N is the representation of word-length of input vectors. The speed of this traditional bit SDA implementation limits its application fields except for certain low speed applications.

Parallel Distributed Arithmetic (PDA) is used to increase the speed performance of SDA. In each clock cycle, the number of bits being processed increases with PDA. The architecture of PDA is implemented by employing more ROM and computing units which work in parallel. Therefore, the number of parallel bits sampled should be increased and the speed of processing is improved hereby. The enhanced performance makes PDA satisfied with the requirements of some high-speed real time application such as image stream cording or decoding.

A typical PDA architecture is shown in Figure III-4.

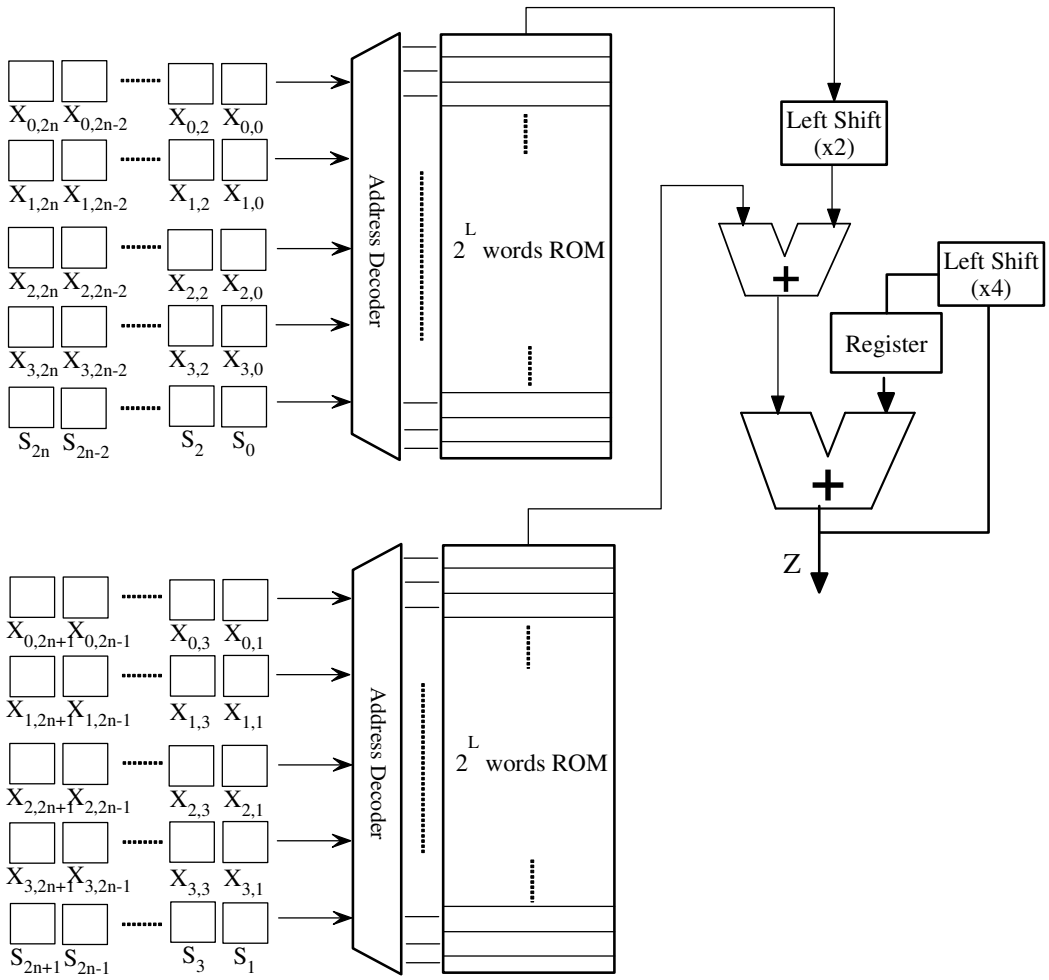

Figure III-4 : A typical parallel DA architecture

The PDA architecture works on the premise that all input vectors are fed parallel to input ports. In PDA architecture, processing speed is doubled compared with SDA by increasing the number of bits processed from 1-bit to 2-bit in half the number of processing clock cycles. Hence, the PDA architecture in Figure III-4 results in as twice the throughput as SDA. By adopting two left-shift registers at half the bit depth, the single serial shift register in SDA is replaced and therefore speeds up the architecture. According to this change, the two parallel memory blocks are employed, one being used for the even-bits and the other for the odd-bits. It is noticed that the cost for 2X speeding up is nearly double hardware occupation. It is a trade-off between speed and hardware cost for a system. In author's opinion, a system

could be twice speeded up at a cost of double hardware except for some systems with lots of redundant parts.

III.4 Applications of Distributed Arithmetic

DA can be found in many applications in fields of multimedia processing and communication. All DA applications involve inner products computation between two vectors, one of which is a constant.

Finite Impulse Response (FIR) can be found in almost all communication systems and digital signal processing. The processing engine of most filter algorithms is a Multiply and Accumulate (MAC) function. DA in filter designs works by distributing the bit arithmetic of the sum of products used to calculate the FIR filter output given in Equation (III-27).

$$Y[n] = \sum_{i=0}^{N-1} h_i X[n-i] \qquad \textbf{(III-27)}$$

where X[n] and Y[n] are the input and output sequences of the filter, respectively; N is the number of TAPs, and i is the i-th coefficient of the filter impulse response. It can be seen from Equation (III-27) that the FIR filtering is based on the MAC function. Filter designs can vary greatly in the number of MACs, from one to thousands. As the number of MACs increases, the algorithm becomes much more complex for general programmable processor architectures. Hence, the algorithm becomes more computing-intensive for any conventional DSP. The MAC function can be implemented more efficiently with various DA techniques than with conventional arithmetic methods.

DCT, one of DA application, is the most widely used algorithms in digital signal processing, which removes artificial discontinuities from highly correlated signals. As one of the major operations in current image/video compression, DCT can be found in JPEG for still picture compression, ITU H.261 and H.263 for video conferencing standard, and ISO MPEG (MPEG-1, MPEG-2, and MPEG-4) for audio, visual compression and communication. Over last several years, significant research work has been carried out on DA techniques and their implementations in DCT and other applications [64-66].

The Discrete Wavelet Transform (DWT) is another DA application, which is one of the most useful and efficient tools used to analyze digital signals in various signal processing areas including compression, signal detection, communications, and time varying spectral estimation. In signal analysis, the DWT has some inherent generic advantages and is nearly optimal for wide class of problems. As a decomposition tool, the DWT separates components of a signal in a way that is superior to most other methods for analysis, processing or compression. This powerful and flexible decomposition tool also offers new nonlinear processing option for signal and image processing, detection, filtering, and compression.

III.5 Conclusion

This chapter has introduced DA concept and its definition first. Two basic DA computations, ROM based DA and adder based DA, are described after summarizing the development of DA concept. What have been presented are the principles of the DA technique and the research directions in the field of image and signal processing. Some architectures based on SDA and PDA have also been described. The advantages of SDA and PDA were also listed in this chapter. Solutions to ROM size problems caused by using DA, have also been proposed. At the end of this section, DA applications in image and signal processing applications and communication are introduced and described briefly.

Chapter IV

Low Power Reconfigurable Architecture for DA

IV.1 Overview

Distributed Arithmetic (DA) has been widely adopted for its computational efficiency in many digital signal processing applications such as DCT, DFT, FIR, and DHT [56]. All these applications involve inner products computation between two vectors, one of which is a constant.

The traditional method to generate products is using a MAC (multiply and accumulate) unit which is fast but suffers from high hardware cost in the case of long-length inner-products. In contrast, DA provides an efficient solution to realize inner product by using memory look-up and accumulation operations. The idea behind conventional ROM-based DA is to replace multiplication operations by pre-computing all possible values and storing them in a ROM. According to [56], a ROM based DA can reduce circuit size by 50-80% on average.

Custom reconfigurable technology has recently emerged to satisfy the demands for both flexibility and efficiency. Custom/domain-specific reconfigurable arrays can be programmed to adapt to different applications, so the efficiency of the hardware and flexibility of the whole system can be improved. Earlier work, such as [41, 67, 68], demonstrates good performance in area, power consumption and speed compared to conventional approach. Since a domain-specific reconfigurable architecture targets at few application fields, it can achieve better performance than a general purpose FPGA device.

In this section, a novel reconfigurable DA architecture is presented which can implement inner products with less area usage and power consumption. The proposed architecture can

support any algorithm for inner product computation, such as DCT, DFT, FIR, and DHT. An adder-based DA, which was introduced in [64], is adopted in the proposed reconfigurable architecture. Compared with a ROM-based DA, the approach needs only 10% of transistor count and 30% of ROM area with comparable performance in the specific application [64]. Our new architecture takes advantage of the common summation terms when the fixed coefficients are decomposed into bit level. It makes the adder array take full advantage of results from the previous level and, therefore, maximizes hardware efficiency. Due to its inherent hardware sharing property, the proposed architecture is suitable for multiple inner product computations. The reconfiguration characteristic makes architecture flexible to switch from one application to another application, which implies that the same hardware architecture can perform different algorithms at different times.

This section is organized as follows. In section IV.2, we review the related work in the literature. The architecture is generally introduced in section IV.3. The algorithm logic unit of the proposed architecture is described in detail in section IV.4 including two-level adder butterfly structure, the Wallace tree multiplier matrix, interconnection network, memory for reconfigure bits and the implementation of these structures. The common terms sharing problem is addressed at the beginning of IV.5, and then the definitions of dimidiate tree and crossing forest are introduced, which is followed by the algorithm developed accordingly for mapping of architecture. Finally, the comparison with subexpression sharing in canonic-signed-digit code is made in sections IV.6.

IV.2 Related Work

Over the last several years, significant research work has been carried out on DA techniques and their implementations in DCT and other applications. In [65], a hardware DA method was implemented for DCT with radix-2 multi-bit coding for minimum resource, and symmetric transpose memory. In [64, 65], an adder-based DA was proposed to generate the inner product of vectors for DCT. The approach reduces ROM area by 70% and transistor count by nearly 90% with comparable performance when compared with ROM-based DA. In [66], a DA-based algorithm is introduced which can formulate 1-D any-length DHT as cyclic convolutions. It simplifies the ROM design process and increases the processing speed for

utilizing identical ROM modules and eliminating the accumulation loop in the processing elements. In [69], a design methodology is introduced which translates high-level compilation algorithmic description of DCT (based on an algorithm by Liu et al. [70]) into a fixed-point, variable-radix, digit-serial dataflow architecture. The methodology allows different designs to be derived from a single algorithmic description and trade-off among quantization effects, throughput, latency and area. However, only serial architecture is used according to the special algorithmic description, which limits the architecture's maximum throughput.

Numerous work has been done on reconfigurable architectures and their implementations. Unfortunately, only one architecture has been found which is designed specifically for reconfigurable DA. In [71], a special reconfigurable architecture for DCT is described. The architecture is designed especially for implementing DCT with different algorithms: pure-RAM, mixed-RAM and COordinate Rotation DIgital Computer (CORDIC), whereas, the architecture could be used only for DCT application, which limits the application fields of the architecture. In [72], a low-power reconfigurable DCT architecture is introduced which is based on efficient trade-off between image quality and computational-complexity. The low-power approach in this paper is based on the modification of DCT base in a bitwise manner with minimum image quality degradation and considerable computational complexity reduction. Various trade-off levels are presented and the reconfigurable architecture can dynamically change from one trade-off level to another without large hardware overhead. The trade-off between image quality and computational complexity and its specific architecture make the method available only for DCT application.

One paper, [73], must be mentioned, which is the first one addressing reconfigurable architecture targeting at distributed arithmetic. This paper presents a domain-specific reconfigurable array targeting at the algorithms that can be implemented using DA. The architecture in [73] is based on ROM-based DA and SDA which has been discussed in detail in previous section and is implemented with two configurable clusters: Add-shift and memory. The elements are arranged in an array with a mesh of reconfigurable interconnects. For different applications, the corresponding coefficients are loaded into memory. Therefore,

the reconfigurability of the array permits mapping a number of distributed arithmetic implementations such as DCT and filtering calculations used in video coding. Just like ROM-based DA and SDA on which the architecture is based, the reconfigurable array in [73] becomes a bottleneck when the outputs data are expected for each clock cycle and the high data throughput is the critical feature for some real-time video streams applications. Another problem with the architecture is that its ROM size grows exponentially as the bit-width of coefficient increases. As the number of inputs and the internal precision become larger, the ROM-based DA suffers from extremely large ROM requirements.

IV.3 Reconfigurable DA Architecture

The proposed reconfigurable DA architecture is shown in Figure IV-1.

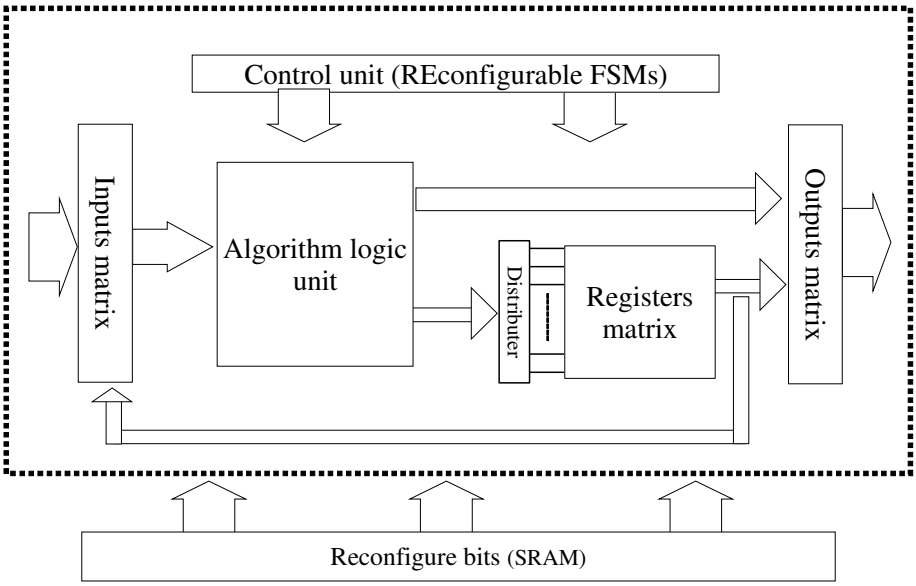

Figure IV-1 : Reconfigurable DA architecture

The architecture is composed of control unit, algorithm logic unit, register matrix, input and output matrix. A SRAM block is closely coupled with the architecture, in which reconfigure bits are stored and will be loaded to the architecture in initialization period. The control unit in the figure is not an independent function module but a virtual module which is made up of the reconfigurable Finite State Machine (FSM) modules existing in algorithm logic unit, register matrix and data paths routing.

In Figure IV-1, the input signals to be processed are sent to the input matrix first. At the same time, the output of the register matrix is sent to the input matrix as well. The input matrix will output the desired data controlled by the control unit. The data from the input matrix will be routed to the algorithm logic unit directly in which input signals are multiplied by coefficient and then accumulated**.** DA operation is done by the algorithm logic unit, which is the reason why the unit is so called. The results from the algorithm logic unit will be routed to the output matrix for export or to distributer which is closely coupled with the register matrix for temporary storing. The results in the register matrix will be sent back to the input matrix controlled by the control unit. Those data will be forwarded to the algorithm logic unit for further processing.

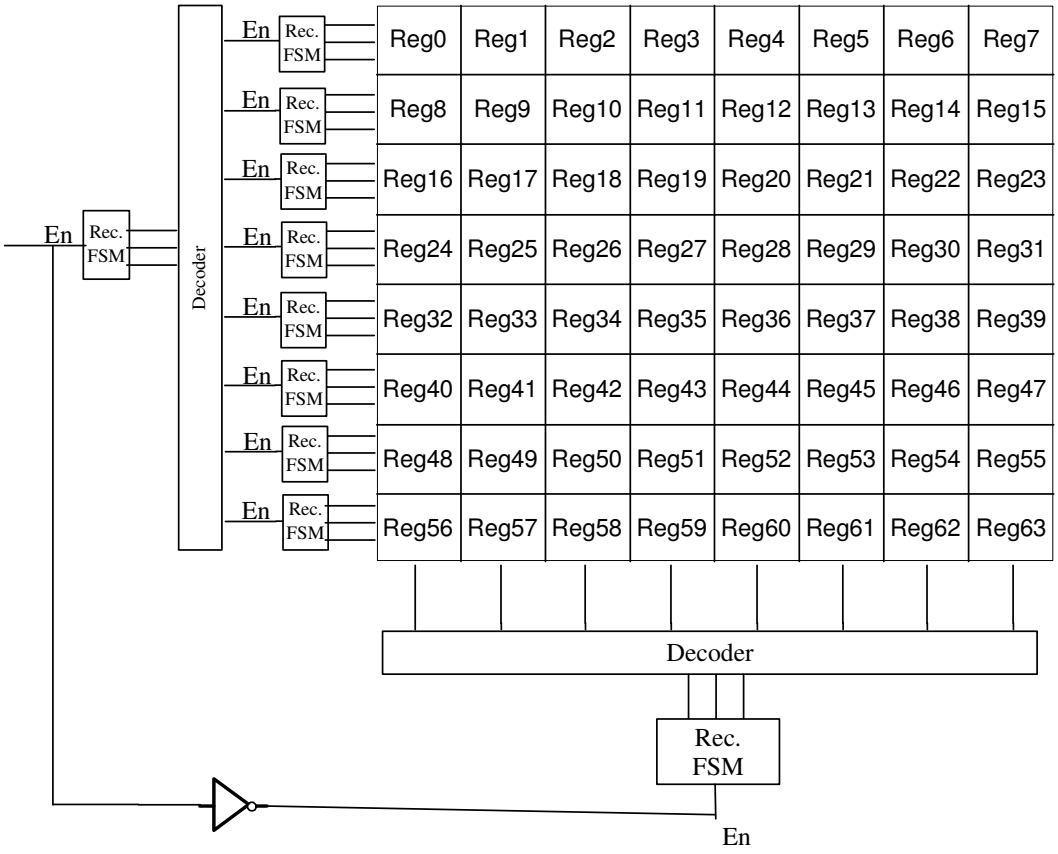

Figure IV-2 : Architecture of address coding

The core processing block is algorithm logic unit which responds to the DA operation. The register matrix is composed of 64 registers which are implemented with flip-flops and can be configured as a one dimension or 8X8 two-dimension array in initialization period. It is used

for temporary data when the architecture is implemented for certain application which exceeds the handling ability of architecture in one clock cycle. The architecture of address coding is shown in Figure IV-2. The module represented by “Rec. FSM” in the figure is reconfigurable FSM module.

When the register matrix works in one dimension mode, the address for total 64 registers can be expressed with 6-bit binary number. In this working mode, only the control units of row take effect, while the control unit for column is disabled. The 6-bit address is divided into two parts: high 3-bit part and low 3-bit part. A 3-bit reconfigurable FSM in the first level will generate the high 3-bit part of the whole address. Through a 3-8 decoder, a 3-bit address can control 8 outputs which are connected to enable control ports of 8 FSMs in the second level, which have 3-bit internal state as well, for low 3-bit part as shown in Figure IV-2. Because of the reconfigurability of FSMs, 64 registers can be reached in any order according to the requirements of applications.

When the register matrix works in 8X8 two-dimension array mode, 64 registers are divided into 8 banks and each bank contains 8 registers. Because the number of banks equals the number of registers in each bank, a register bank can be a row or a column. In this work mode, 8 registers in a bank can be read/stored in parallel at each time. To address 8 banks, a 3-bit binary number is used which is generated by a 3-bit reconfigurable FSM. As shown in Figure IV-2, two FSMs address the banks in row and in column respectively and the enable ports of FSMs are inphase opposition by connecting an inverter between them. This mechanism, the invert, is only available in 8X8 two-dimension working mode and makes two FSMs working in different time and ensures that the two working states, row and column, of the register matrix will not operate at the same time. In row working state, the FSM in the first level is enough for 8 addresses coding and the 8 FSMs in second level in row control unit are therefore disabled.

It can be seen from the work flow that control unit makes the data stream ordered and drives the data to the desired destination according to the specific application when it is configured in initialization period. The mechanism of the control unit is a multi-input-multi-output FSM which can be programmed and configured as a specific FSM to implement the controller

output functions and next state logic. Due to its reconfigurability, the control unit can be made arbitrarily complex and can realize any control function under its capability. Being such an important function unit in the architecture, its architecture will be discussed in Chapter V. Its performance in area, power and delay will be evaluated and analyzed in Chapter V as well.

The algorithm logic unit is the core processing unit which realizes the multiplication between input signal and coefficient vectors without multiplier. It can be reconfigured to implement certain DA application according to the corresponding configure bits which are pre-stored in external SRAM. More details of algorithm logic unit will be given in this chapter including the architecture of sub-function units and its mapping algorithm.

IV.4 Architecture of Algorithm Logic Unit

The architecture of algorithm logic unit in proposed reconfigurable DA architecture is shown in Figure IV-3.

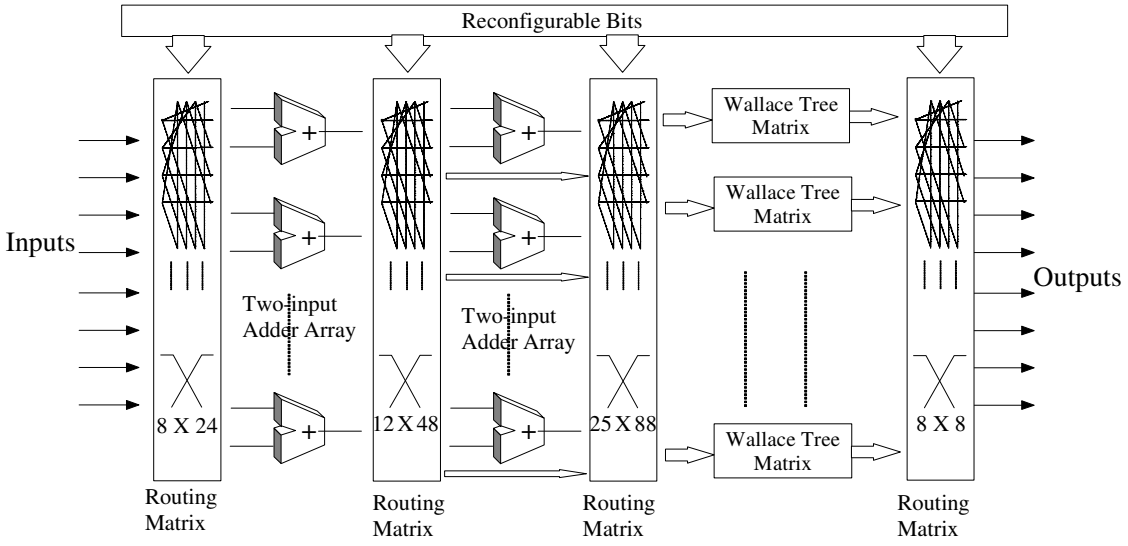

Figure IV-3 : The architecture of algorithm logic unit

It consists of two major parts: the first part is two-input adder arrays in two levels which realize T_j term as defined in Equation (III-10), and the second part is parallel Wallace tree multiplier matrix which generates the final results Z. The number of input of the system is up to 8 with 9 bits width.

For some simple cases as listed in Figure IV-4, only one level of common terms sharing is adopted, which is enough for a simple case. For more complicated cases such as DCT, DFT, an adder structure with two or more levels would be better for reducing the hardware area, but at the cost of extra delay time. The more levels, the longer the delay time. Actually, a structure with three or more levels of common terms sharing would make the whole architecture very complicated in term of interconnection and routing network. To balance the area usage and delay time, a two-level adder structure has been adopted in this work.

IV.4.1. Algorithm of Proposed Architecture

The adders with shifts replace the multipliers in the original DA algorithm. The adoption of adders also makes the architecture more hardware efficient. However, the benefit of adoption of adders is not limited to hardware efficiency; its speed also achieves N times as fast as ROM-based DA, where N is the bit-width of the input vectors. In ROM-based DA, the vectors are imported serially to generate the ROM addresses for computing R_k terms, while in adder-based DA, all inputs are fed parallel to the adders for computing T_j. The time is consumed only by adders; the wider the inputs, the more time is taken.

Besides the advantages of adder-based DA described above, common terms sharing brings additional advantages by further reducing hardware complexity. Figure IV-4 shows an example case for the proposed approach.

We have term T_j, defined in Equation (III-10), which is

$$T_j = \sum_{i=0}^{L-1} X_i C_{i,\,j}$$

Suppose the input vector and fixed coefficient vector are

$X = [X_0, X_1, X_2, X_3]$

$C_{00} = 1101_b$, $C_{10} = 1011_b$, $C_{20} = 1110_b$, $C_{30} = 0011_b$;

Then the expanded form of T_j is shown in Figure IV-4.

$$
\begin{aligned}
T_j = \quad & X_0\,(1\ 1\ 0\ 1) \\
& X_1\,(1\ 0\ 1\ 1) \\
& X_2\,(1\ 1\ 1\ 0) \\
+) \quad & X_3\,(0\ 0\ 1\ 1) \\
\hline
& X_0 \quad X_0 \quad 0 \quad X_0 \\
& X_1 \quad 0 \quad X_1 \quad X_1 \\
= \quad & X_2 \quad X_2 \quad X_2 \quad 0 \\
+) \quad & 0 \quad 0 \quad X_3 \quad X_3 \\
\hline
& = (X_0+X_1+X_2)\,2^3 + (X_0+X_2)\,2^2 \\
& \quad + (X_1+X_2+X_3)\,2^1 + (X_0+X_1+X_3)\,2^0
\end{aligned}
$$

Figure IV-4 : Example of adder-based DA

In Figure IV-4, C_i and X_i are substituted in expression of T_j first, and then fixed coefficients are decomposed into bit level, each bit corresponding to different power of two. After multiplying input vectors and their corresponding coefficient in bit level, input vectors can be found in the position where their respective coefficient bits are ones as shown in Figure IV-4. Obviously, the vectors in different positions indicate the different power of two. The final result can be obtained by adding all vectors with their corresponding power of two. Notice that the additions are taken only at the nonzero bits of coefficients. From the expression of final result in Figure IV-4, one finds that there are some common terms between different bit weights: term X_0+X_2 between bit weights 2^3 and 2^2, term X_1+X_3 between bit weights 2^1 and 2^0.

IV.4.2. Two-level Adder Structure

In the two-level adder butterfly structure, the first level consists of up to 12 parallel 9-bit adders. The inputs are fed to this level through a routing matrix. There is a bypass path followed by every adder in this level, which allows the input data pass straight to the input ports of second level adder array. The 12 output ports of first level are routed to the next level inputs by another routing matrix. Due to the common terms sharing, one output in current level can be shared by two or more adders in the following level. The second level adder array consists of up to 24 parallel 10-bit adders. Compared with the first level adder array, each adder in the second level is followed by a 2-input multiplexer, as shown in Figure IV-5, which allows the outputs of the first level adder array to pass straight to the Wallace

tree multiplier matrix when the application can be implemented with only one level common terms sharing. The bypass paths in two levels adder array make input data routed directly to Wallace tree multiplier matrix. The bypass path makes the architecture more flexible for target application by switching between one or two-level adder structure.

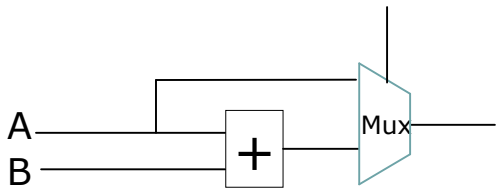

Figure IV-5: Adder followed by 2-input multiplexer

The two-level adder structure with 9-bit and 10-bit adders can generate the summation of 2 or 4 inputs with 8-bit. Besides, a special 11-bit adder is set for the summation in the second adder array for the case of 8 inputs addition. The output of two-level adder butterfly will be fed to the parallel Wallace tree multiplier matrix through routing matrix.

IV.4.3. Wallace tree multiplier Matrix

After one or two-level adder structure, all outputs will be routed to the parallel Wallace tree multiplier matrix to generate the final outputs *Z*. A serial addition approach is the easiest and most straightforward way for obtaining *Z*, in which N cycles are needed to complete the accumulation where N is the precision of coefficient C_i. Therefore, the delay incurred by a serial addition approach cannot fully satisfy the requirements of some real time applications. In view of the adder array, a serial addition approach will never take full advantage of the parallel structure available. With serial addition, only one input is added at each cycle. Obviously, a serially computing model does not make full use of the previous level hardware and results.

To avoid timing bottleneck and inefficient use of hardware, a parallel processing approach is adopted in the second part. A structure with 8 parallel Wallace tree multiplier blocks provides 8 outputs at once. A traditional Wallace tree multiplier and 3:2 compressors are used for each accumulation.

The Wallace tree multiplier, an adder tree built from carry-save adders, is one of the well-known methods for speeding up the accumulation of data with large word length and where the performance is critical. It is considerably faster than a simple array multiplier because its height is logarithmic in word size. A Wallace tree is a bit-slice adder which adds all the bits in the same bit position. The principle of Wallace tree multiplier is shown in Figure IV-6.

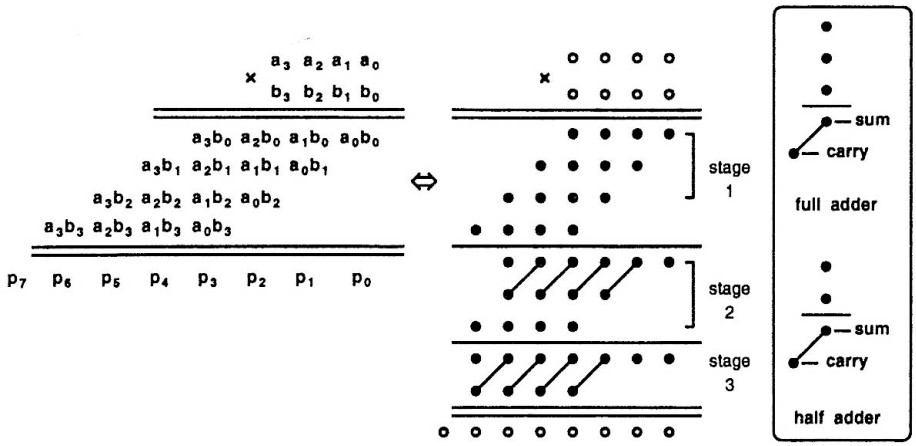

Figure IV-6: Operation of the Wallace tree multiplier [74]

According to the figure shown above, 16 partial products need to be summed up. The first step in the addition process is to group three rows of partial products with full adders for three dots in one column and half adders for two dots in one column. The results from the full and half adders are passed on to the second stage. The process of grouping partial products in stage 2 and 3 with full and half adders adopt the same rules as used in Stage 1. Finally, the product of the two 4-bit multiplication is obtained. Because summations of full and half adders at each stage is done in parallel, the time required for the multiplication is the delay at each stage multiplied by the stage number. Therefore, the Wallace tree multiplier and 8 parallel Wallace tree multiplier blocks are adopted in the proposed architecture to achieve the fastest processing speed.

IV.4.4. Interconnection Network

As discussed in previous chapters, reconfigurable processors show their attractive characteristics including both high-performance and energy-efficiency in a domain-specific application compared with general purpose reconfigurable architecture. The proposed architecture in this dissertation presents such a reconfigurable solution by combining pre-

designed functional units through a reconfigurable interconnect network. The routing matrix in algorithm logic unit is one part of processor interconnect network, which connects an arbitrarily chosen subset of inputs with desired output ports. It provides powerful connectivity between functional blocks and flexibility for signals routing. Generally, high flexibility comes with high energy consumption. The objective of creating routing matrix is to maximize the routability of the interconnection network at the possible lowest power consumption.

In this sub-section, various reconfigurable interconnect schemes and their implementations are introduced. Based on a detailed comparison of the presented reconfigurable interconnection, our interconnection matrix is described including concept and implementation.

A. Full Crossbar

A full crossbar interconnect allows simultaneous connections from any input port to any output port. An NxM crossbar consists of N parallel input wires and M parallel output wires; they are placed orthogonally so that each input wire crosses every output wire, and programmable switches are placed at all cross-points to join the pairs of wires. In a full crossbar, a switch joins each pair of input and output wires [75]. Figure IV-7 shows a diagram of a full 8 x 6 crossbar where the vertical wires are inputs and the horizontal wires are outputs.

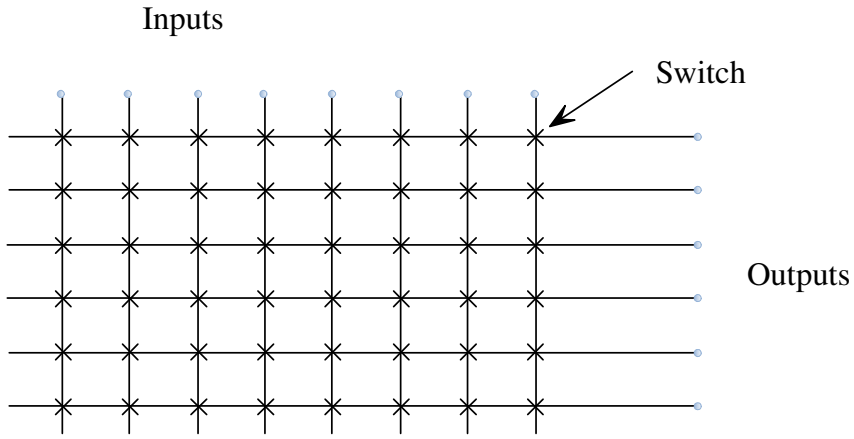

Figure IV-7 : An 8x6 full crossbar

A crossbar can be viewed as a 2-sided switch block: the input wire terminals are on left side and output wire terminals are on the other side, and two terminals are joined by a switch if and only if there is a switch joining the two wires of the terminals in the crossbar, as shown in Figure IV-8.

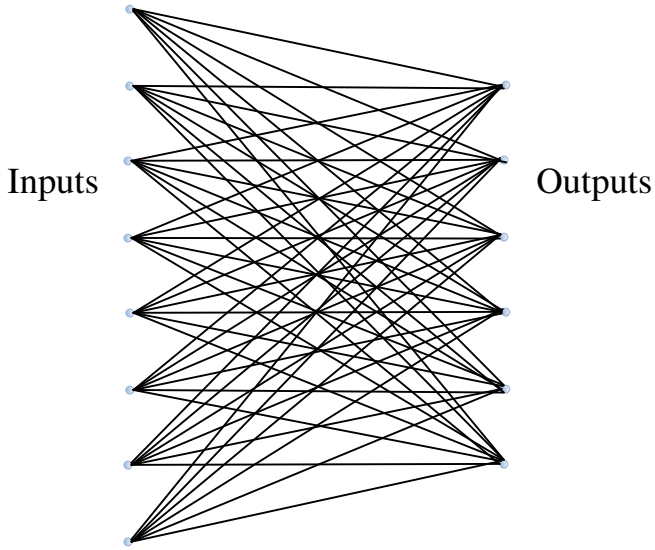

Figure IV-8 : 2-sided switch block

It is clear that a full NxM crossbar with $N \leq M$ can route an input signal to an N-subset of output wires in any given order.

B. Multiple-bus

Multiple-bus can be viewed as an extended implementation of full crossbar when input data is not only sent to output ports. The objective of multiple-bus is to share input signals with more integrated modules. It can be implemented by modifying outputs in full crossbar as buses and adding a number of vertical wires as outputs. See Figure IV-9 for an example.

An NxBxM multiple-bus consists of N parallel input wires, B bus wires and m output wires. Programmable switches are placed at all cross-points to connect the pairs of wires. Each selected input signal is routed to the target bus wire first by conducting the switch between input and bus wire. Output ports can get their desired data by controlling the switch between the output wire and corresponding bus wire.

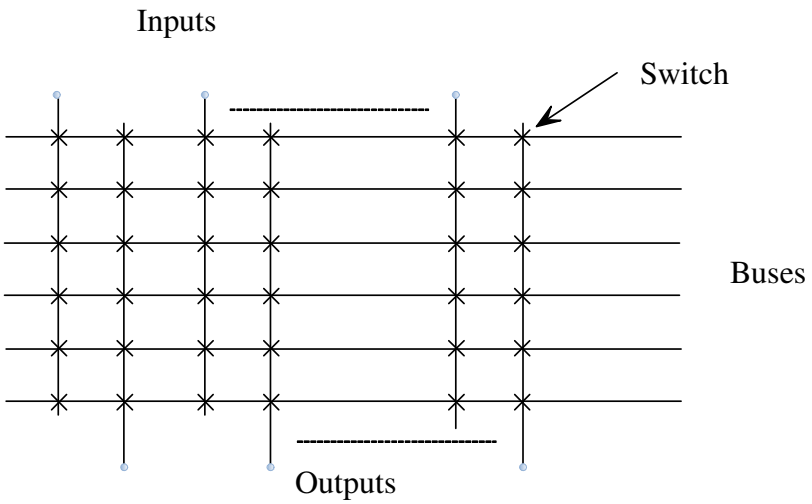

Figure IV-9 : Multiple-bus interconnection

Obviously, an NxBxM multiple-bus with $N \leq B \leq M$ can route an input signal to an N-subset of output wires in any given order in two stages: data being routed first to bus wires and then output ports.

C. Multi-stage Interconnection

An alternative design approach of routing matrix is based on multi-stage interconnection networks. Multi-stage interconnects were introduced and discussed in detail by Clos in [76] and Benes in [77]. The idea behind multi-stage interconnection networks is using longer delay time to reduce the complicity of hardware compared with full crossbar. A multi-stage interconnection network is a non-blocking network, which can provide a connection path from any idle input and output without interference with other active connections. Moreover, the setup of a new path between specified input and output ports will not affect the possibility of future connection requests.

The multi-stage network follows distributed routing. Its control units are distributed among switches in several levels. A route path from input port to output port is separated into several segments, not a straightforward one as used in crossbar. Each segment has different intricate interconnect patterns at their respective switch levels, so for a given source or destination address, the routing decisions can be made independently according to routing strategy of the switch. All the data from input ports are routed to the first level switch

controllers and then the data will be sent to the second level switch controllers. Just following this method, the data will finally reach the destination output ports.

In the whole routing path, routing decision is made at each individual switch. This greatly reduces the complexity of the switch controller. In general, the number of switches can be reduced from N^2 to NlogN. Also, independent switches make it possible to alter the switch configuration without disturbing others. Another advantage of distributed routing is its easy-to-control connection. It also makes the routing time equal to the propagation delay of the switches. Therefore, latency introduced by the switch controller is removed owing to the reduced complexity of the network. The number of intermediate stages grows with the number of inputs and outputs.

Multi-stage network includes omega networks, delta networks and many other types. A conceptual view of an omega network is shown in Figure IV-10.

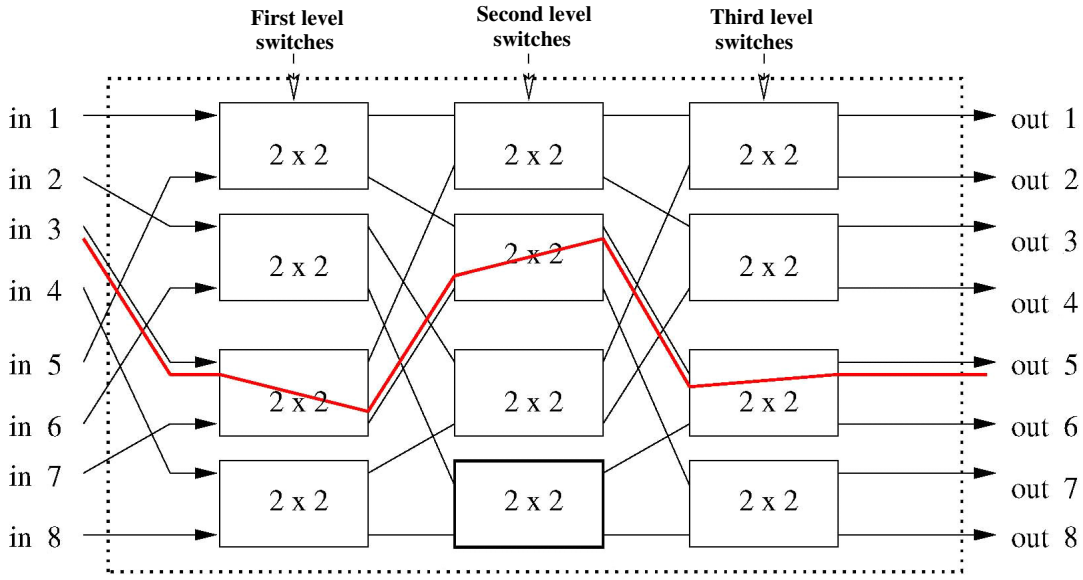

Figure IV-10 : An omega network [78]

D. Comparison between Implementations

A full NxM crossbar has N·M switches which means M switches are required for each output. In multiple-bus construction, B·(M+N) switches are used in total. For a single output port, (B + B·N/M) switches are necessary. Regarding multi-stage network, it adopts MlogN

switch to build the whole architecture. For each output port, logN switches are used. It can be concluded that multi-stage network is most hardware-efficient from the view of output port. Multiple-bus construction is less efficient in hardware than multi-stage network, but it provides extra data delivery ability compared with other modules. A full crossbar is the least hardware-efficient in three implementations.

It can be observed that a full crossbar network requires only one switching stage, that is to say, every input and output pair is connected through a single switching element. This architecture provides full connection flexibility, but suffers from large area overhead as we have discussed above. In multiple-bus construction, input data will reach output port after two switching stages, the double of that in crossbar network. The slowest one is multi-stage network among the three. It requires $\log_2 N$ switching stages for the whole signal travelling path, a lot more times than that of crossbar network. From the view of latency, a full crossbar is the most efficient one. But it also suffers from its high energy consumption due to the long global buses and the large number of switches.

The comparison indicates a general rule in design that it is a trade-off relationship between area, power and delay. Without the revolution in technology, the way to improve the speed of architecture will be adopting more hardware to shorten crucial patch and hardware efficiency will be improved at the cost of extra delay time.

E. Proposed Interconnection Network

The idea behind the design of interconnection network in our processor is to balance the performance of area, power and delay and make the processor satisfy the application requirement.

Because the system will be potentially used for high speed real-time image processing, the slowest scheme, multi-stage network, in three implementations is out of our sight although it is the most efficient in area. Full crossbar is the fastest one as we discussed above, but it is too expensive when N and M are large. But crossbar will work if it can be simplified and the number of switches can be cut down based on the targeted applications requirement. The problem of designing crossbars with large N, M focuses on selecting cross-points where

switches are going to be placed to satisfy certain routing specifications. There are several problems in designing such crossbars. One of them is the so-called sparse crossbar design problem [79, 80] that in designing a partial N x M ($N \geq M$) crossbar with a linear number of switches in terms of n, the percentage of routable routing vectors is indicated by the given switch count. This percentage is so-called routability which is defined as the likelihood that an arbitrarily chosen subset of inputs can be connected to outputs. A routing vector is an n-dimensional 0-1 vector (x_1, x_2,……, x_n), which is used to represent a selection of input terminals with $x_i = 1$ meaning that the i-th input terminal is selected. A routing vector is routable if all the selected terminals can be routed to output terminals simultaneously.

Another problem in designing is how to build a partial N x M ($N \geq M$) crossbar with a minimum number of switches so that every group of N inputs can be routed to M outputs. Nakamura and Masson [81] showed that an optimal design has $(N - M + 1) \cdot M$ switches. Based on the existing crossbar theory and algorithm, several optimal crossbars are designed according to the routing requirement. One of the optimal crossbars is shown in Figure IV-11 as an example which has 8 input ports and 6 output ports.

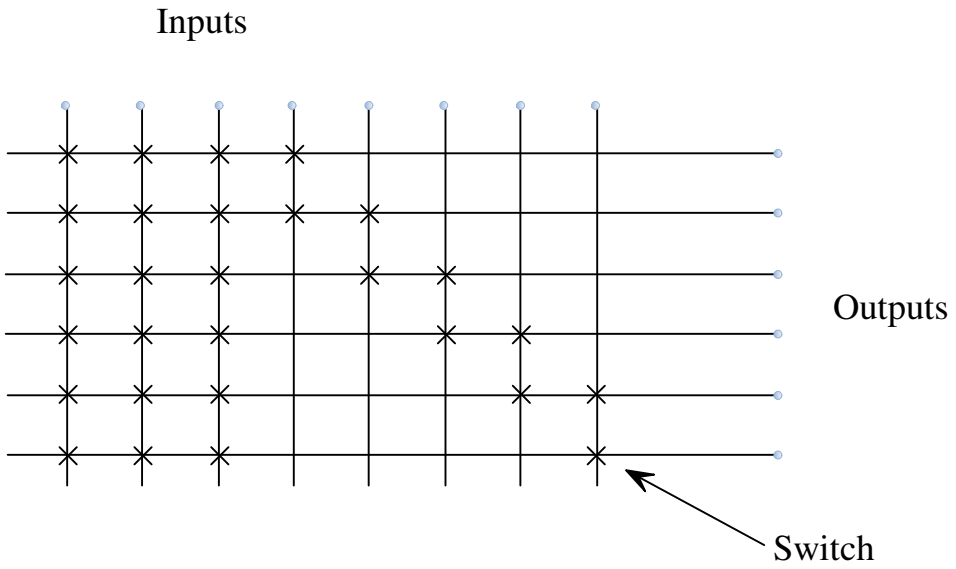

Figure IV-11 : A partial 8x6 crossbar

It can be viewed as a 2-sided switch block as shown in Figure IV-12;

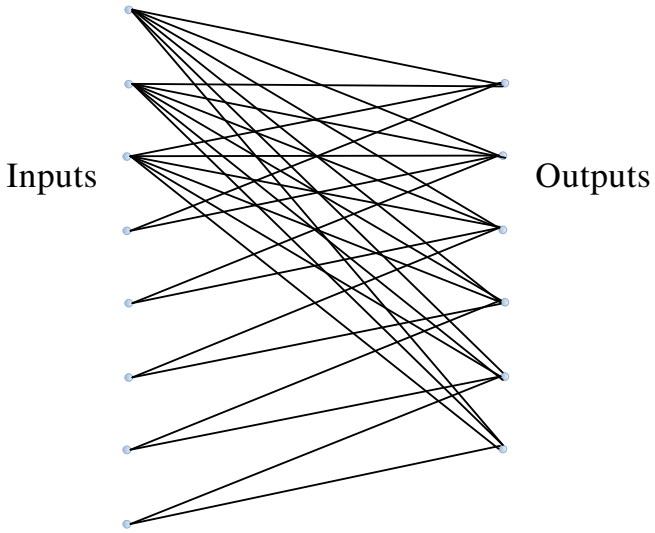

Figure IV-12 : A 2-sided switch block for partial crossbar

IV.4.5. Memory for Reconfigure Bits

For different field-programmable devices, different approaches to implementing programmable switches have been developed. For commercial CPLDs the main switch technologies are floating gate transistors like those used in erasable programmable read-only memory (EPROM) and electrically erasable programmable read-only memory (EEPROM), and for FPGAs they are SRAM. Each of these will be briefly discussed below.

EEPROM and flash memories retain their contents because they use floating-gate transistors: application of extra high or low voltage differentials across a floating gate leads tunnelling to deposit or remove electrons from the floating gate, changing the threshold of the EEPROM or flash transistor. Regular operating voltages have negligent effects upon these floating gates, just as the absence of power does not affect them, so they are able to retain their programmed states even when powered off.

An EEPROM/EPROM transistor is adopted as a programmable switch in PLD/CPLD. They are placed between two orthogonal wires acting as a controlling transistor to isolate or connect two lines, as shown in Figure IV-13.

In the figure, two wires are connected by a transistor, with the gate of the transistor being controlled by an EEPROM/EPROM cell. By setting the bit to 1, signals can be driven across the transistor. When setting the bit to 0, the horizontal and vertical wires are isolated.

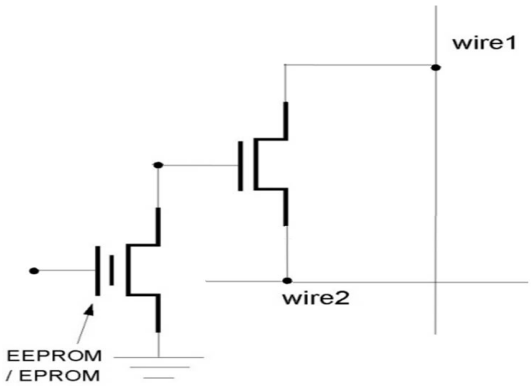

Figure IV-13: EEPROM/EPROM programmable switch

One constraint on our designs is that we must implement our reconfigurable architecture in the same process technology that the rest of the SoC is going to use, since everything will be implemented on a single piece of silicon. While floating gates are useful in creating certain memories, they are not particularly useful in the design of processors, DSPs, or custom logic, and it is these other blocks that will dictate the process technology of the SoC. EEPROM and flash memories are therefore not reasonable memory choices for us, because the SoC fabrication process will not have the ability to create the necessary floating gates.

For technical reason, EPROM or EEPROM could not be applied to FPGAs to obtain a high density. Currently, commercial FPGA products are mainly based on SRAM technologies. SRAM memory cells, on the other hand, are created without the need for floating gates. A simple 6-transistor SRAM cell is shown in Figure IV-14.

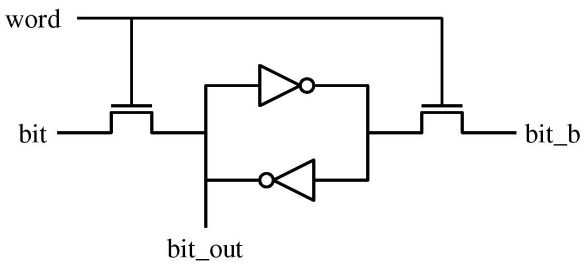

Figure IV-14: A simple 6-transistor SRAM cell

The function diagram for SRAM looks similar to EEPROM/EPROM cell. SRAM cells between wires can be programmed to connect or disconnect the wires according to the required logic function. Wires and configurable switches like the one shown in Figure IV-14 can be easily used to create a full crossbar. In the normal operation of a crossbar, each output wire is connected to exactly one input wire, with the numbers of output wires less than or equal to the number of input wires, as shown in Figure IV-15.

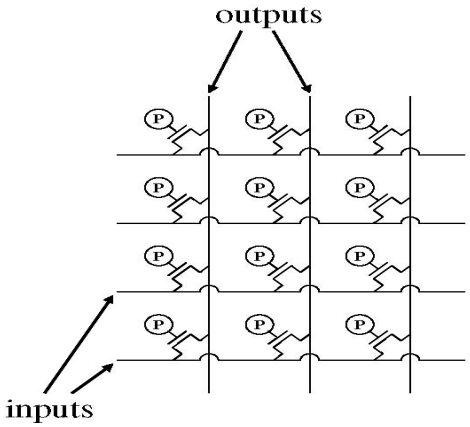

Figure IV-15: SRAM switch based full crossbar

We now have several horizontal wires each of which can be connected to any of several vertical wires. These orthogonal wires build up a switch matrix which can route any input signal to the desired output port. An example of usage of SRAM-controlled switches is illustrated in Figure IV-16.

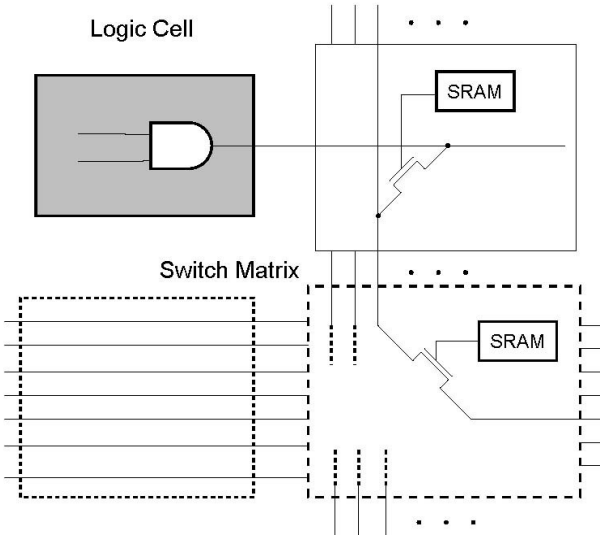

Figure IV-16: SRAM programmable switch application

In this figure, SRAM works as pass-transistor switch to control the connection between different gate nodes. As an example shown in the figure, the output signal from one logic block in the upper left corner is switched to another logic block in the down right corner through two pass-transistor switches in switch matrix. The SRAM cell between the appropriate wires is set to conduct the two orthogonal wires to realize the connection.

The SRAM based switch can be fabricated using the same technology that most SoC devices will use, and is therefore an attractive memory solution for our reconfigurable architectures. Thus, SRAM will be used as the configuration memory in our work.

All reconfigure bits are stored in the external SRAM block. For different applications, the corresponding reconfigure bits are loaded into SRAM. When the architecture is targeted at certain application, the pre-stored data in SRAM will be loaded to the architecture to configure the functional units first and then all registers and logical blocks are reset to original value. When all works above are done, the architecture has finished its initialization proceeding and ready for the targeted application.

IV.4.6. Architecture Implementation:

The architecture was implemented using the Verilog hardware description language. A standard-cell based synthesis and layout was performed with Design Compiler from Synopsys, Inc., targeting at the UMC 0.18 μm CMOS technology library.

All the design is provided as synthesizable soft-core without the specific custom library to allow the customization of all the aspects of the array at design-time. This also permits an easy integration of the arrays into the SoC architecture and software flow.

To evaluate the performance of architecture, DCT and FFT algorithmic are implemented in sections VI.1 and VI.2 respectively. The power consumption of our reconfigurable architecture was obtained after post-layout simulation by the Synopsys PrimePower. The experimental results and analysis will be given in sections VI.1 and VI.2 as well.

IV.5 Algorithm Searching for Optimal Scheme

As discussed in section IV.4.1, the sharing common terms could save the hardware area and reduce the redundant computing to save power consumption. One will find that the diverse common term schemes will result in different amount of resource savings. For the example shown in Figure IV-4, seven two-input adders are needed for accumulation without sharing common terms. There are three common term sharing schemes: X_0+X_1, X_1+X_2 and X_0+X_2 & X_1+X_3. In the first two schemes, six two-input adders are needed. In contrast, five two-input adders are needed for the third scheme. Therefore, common terms sharing will make the system area efficient and power-saving.

It is noted that the characteristics of common terms sharing are available in some cases as discussed above. To make the architecture take full advantages of common terms sharing, parallel multi-output construction is adopted which processes and outputs data for several output ports simultaneously. This mechanism makes it possible to share common terms between different output ports, not just limited to an output port itself. It will greatly improve the utilization ratio of common terms. Currently, we adopt a structure with 8 parallel Wallace tree blocks providing 8 outputs at one time**.**

Only the best scheme of common terms sharing will take full advantage of sharing characteristic and maximize hardware efficiency in the implementation. The way to find out optimal scheme will be discussed in detail in the following sections.

First, we will discuss the availability of common term sharing which is fundamental and the most important issue of all. The whole architecture will become less meaningful if it is available only for some special cases, in which common terms sharing is one of the theoretical bases. To solve the problem of optimal scheme, the related definitions and analysis in the mathematics field are given and a visual tree is adopted to mathematically model our proposed architecture. The definitions of the related concepts in dimidiate tree and crossing forest are introduced to describe the issues under consideration. Then, an algorithm for searching for the optimal scheme is developed and analyzed.

IV.5.1. Common Term Sharing Availability Analysis

Common term sharing characteristic will make the architecture, with which the application is implemented, consume less power and reduce hardware occupation, although the functionality of the application's implementation with the architecture will not be affected even without applying common term sharing.

The Figure IV-4 is a special case. Then, is the common terms sharing property available in real applications? Let's consider random constant vectors and take 4-input-4-bit coefficient case as an example. The coefficient of such case is a matrix which is four lines with 4 elements on each line. Totally, there are 16 elements and 65536 (2^{16}) cases. Take the number of '1' element as the index to analyse the possibility of applying common term sharing characteristic.

The table below is used to analyse the possibility of applying common term sharing characteristic. Numbers of '1' are filled in the first column, making index 1 to 16. For index of 16, 15, …, 10, 9, 8, …, 2, 1, 0, there are totally 1 ($=C_{16}^{16}$), 16 ($=C_{16}^{15}$),…, 8008 ($=C_{16}^{10}$), 11440 ($=C_{16}^{9}$), 12870 ($=C_{16}^{8}$), …, 120 ($=C_{16}^{2}$), 16 ($=C_{16}^{1}$), 1 ($=C_{16}^{0}$) cases respectively, which is listed after the index number in the second column. Among the total cases, some satisfy the requirement of common term sharing. The number of these cased is demonstrated in the third column as "available cases" and the percentage of them in total cases is shown in the last column.

Taking index 10 as an example, there are six zeros in the coefficient matrix. Among these cases, only the cases with four zeros located at diagonal have the possibility to disable the common term sharing characteristic. The number of such cases is 36. By exhaustive search, only 6 cases cannot be applied with common term sharing, so 8002 (=8008-6) cases satisfy the requirement of common term sharing. So, for index 10, as shown in the table, there are totally 8008 cases. Among them, the ones satisfying the requirement of common term sharing amounts to 8002 and takes up 99.9% of all. Similarly, index 9, 8, 7, 6, 5, 4 have 11176, 12480, 7200, 2304, 432 and 36 cases respectively that are fully fit for the sharing characteristic. For cases in index 16, 15, 14, 13, 12, 11, common terms sharing can be applied without any exception.

Table IV-1: Availability analysis of common term sharing

Index	Total Cases	Available Cases	rate (%)
0	1	0	0.0%
1	16	0	0.0%
2	120	0	0.0%
3	560	0	0.0%
4	1820	36	2.0%
5	4368	432	9.9%
6	8008	2304	28.8%
7	11440	7200	62.9%
8	12870	12480	97.0%
9	11440	11176	97.7%
10	8008	8002	99.9%
11	4368	4368	100.0%
12	1820	1820	100.0%
13	560	560	100.0%
14	120	120	100.0%
15	16	16	100.0%
16	1	1	100.0%

To sum up index 1 to 16, 48515 cases from totally 65535 cases are available for common terms sharing property. The effectiveness ratio for 4-input-4-bit coefficient cases is 74% and it will rise as the number of input and coefficient bit width increases.

The ones and zeros in coefficient matrix contribute quite differently even their numbers are the same. Taking index 12 and 4 as examples , for all the cases of the coefficient matrix with only four ones, 36 cases are available for common terms sharing; while for all cases with only four zeros in coefficient matrix, all of them satisfy the sharing characteristic, without exceptions.

Therefore, in theory, the cases with at least one common term are more popular compared with those with none. For real application such as DCT and DFT, the coefficients in the applicable applications are symmetric and regular and this makes the application take full advantages of common terms. An example for DCT will be introduced in the next chapter, which will demonstrate that the common terms sharing contributes greatly to the resource savings.

IV.5.2. Dimidiate Tree

According to the characteristics of the adder array, two inputs from its previous level and one output for its next level, dimidiate tree and its properties are naturally utilized to visually define and describe the architecture. It should be noted that the tree defined here is similar to the binary tree in the computing related data structures. The reason for defining the new concept-dimidiate tree is that some properties of binary tree cannot be applied in our case. To avoid confusion and misunderstanding on the problem addressed and architecture proposed, dimidiate tree and its properties are introduced in this sub-section. Though being a newly defined concept, the dimidiate tree can be considered a special case of binary tree.

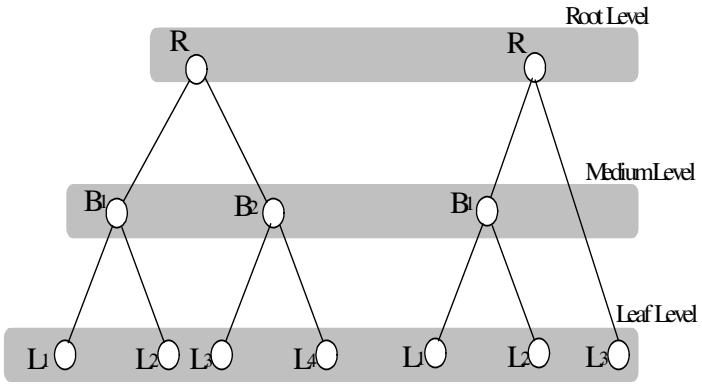

Figure IV-17 : Examples of dimidiate tree

Tree presentation arises naturally based on the hardware architecture. The relationships between inputs of architecture, outputs of the first level adder array and output of the second level adder array can be presented as a tree, shown in Figure IV-17 with symbol *L*, *B* and *R* respectively. A tree is a collection of elements called nodes, one of which is distinguished as a root, along with a relation that places a hierarchical structure on the nodes. Each node in the tree denotes the summation of certain inputs with number ranging from one to four in our case. Generally, a tree can be defined recursively in such a manner: suppose n is a node and $T_1, T_2, \ldots, T_k$ are trees with roots $n_1, n_2, \ldots, n_k$, respectively. A tree can be constructed by making n the parent of nodes $n_1, n_2, \ldots, n_k$.

In Figure IV-17, R is the root of the tree and $B_1, B_2, L_1, L_2, L_3, L_4$ the subtrees of the root. Nodes $B_1, B_2, L_1, L_2, L_3, L_4$ are called the children of the node R. The parent-child relationship is depicted by a line as shown in Figure IV-17. This tree is defined as dimidiate

tree each node of which has either no child or exactly two children without order. This characteristic of dimidiate tree accords with the two-input adder in the architecture.

If $n_1, n_2, \ldots, n_k$ is a sequence of nodes in a tree and n_i is the parent of n_{i+1} given $1 \leqq i < k$, then n_i is an ancestor of n_{i+1}, and n_{i+1} is a descendant of n_i. If a proper ancestor or proper descendant is defined as an ancestor or descendant of a node, rather than the node itself, as shown by the tree in Figure IV-17, the node R is the root which is the only node with no proper ancestors. In contrast, the nodes L_1, L_2, L_3, L_4 are called leaves for they have no proper descendants.

In our case, the leaves in a tree are related to the inputs of architecture, and the roots of a tree then represent the outputs of two-level adder array**.** Three levels are defined to illustrate different node types, root level, medium level and leaf level as shown in Figure IV-17. All the nodes with the same property are located on the same level. At root level, nodes are the root of each tree and represent the summation of three or four inputs. All nodes for the summation of two inputs are located at medium level; while leaf level, the bottom level of the tree, contains all single inputs. It is noted that the nodes location arrangement is different from the one in binary tree. The most distinguished difference between dimidiate tree and binary tree is the definition of level and its characteristic. The nodes in binary tree are generally located in accordance with the parent-child relationship between two nodes. Correspondingly, nodes in our dimidiate tree are arranged in their corresponding level strictly based on their presentation.

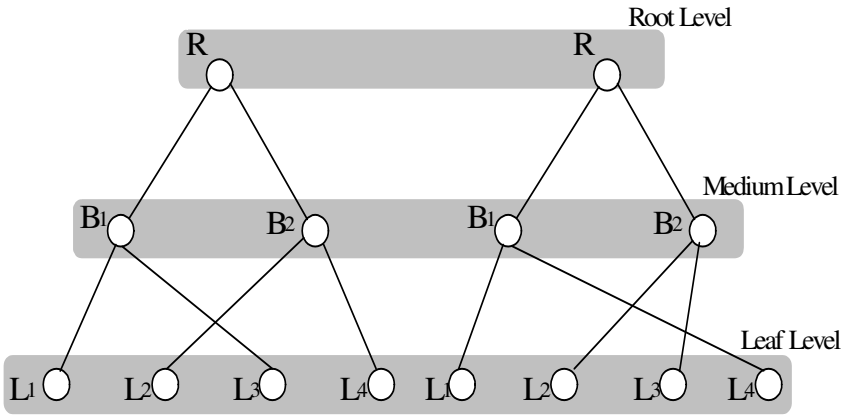

Figure IV-18 : Different format for the same dimidiate tree

Therefore, the most important characteristic of node is which level it belongs to. In dimidiate tree, the relationship between the nodes of two levels is fixed but the specific parent-child relationship between the two nodes can vary, which means that the format of a unique dimidiate tree is versatile. Taking the first tree in Figure IV-17 as an example, the other two formats of the tree are shown in Figure IV-18.

From the Figure IV-18, the node B_1 can be the parent of L_1, L_2 pair, L_1, L_3 pair and L_1, L_4 pair. The parent-child relationship is never changed no matter which specific relationship is selected. In the definition of dimidiate, a tree is exclusive when the levels of all nodes of the tree are determinate. The tree is disordered for the order of nodes in each level is inessential for distinguishing one tree from another. According to this property, we can obtain a useful inference for further discussion, called property of uncertainty, that is

> ∗ *In a dimidiate tree, the undetermined nodes in medium level are not unique even when their ancestor or descendant nodes are known.*

A dimidiate tree indicates the structure of two-level adder array for one output. This tree representation is established by merging two children nodes to a parent node. With the definition of tree, the targeted problem can be addressed and described clearly.

IV.5.3. Crossing Forest and Targeted Problem

The purpose of defining the dimidiate tree is to address the problem of common terms sharing scheme discussed at the beginning of section IV.5 in this chapter. Selecting different schemes will greatly affect the hardware efficiency and, thereby, power consumption.

Features of certain applications makes it necessary to consider the case that multiple trees are put together. For the reason of the repeated nodes, one child node may be shared by different parent nodes, that is, one child node can have more than one parent. If n denotes the number of parent nodes, the child nodes of dimidiate tree will be less than $2n$. The example in Figure IV-19 gives an illustration of this case.

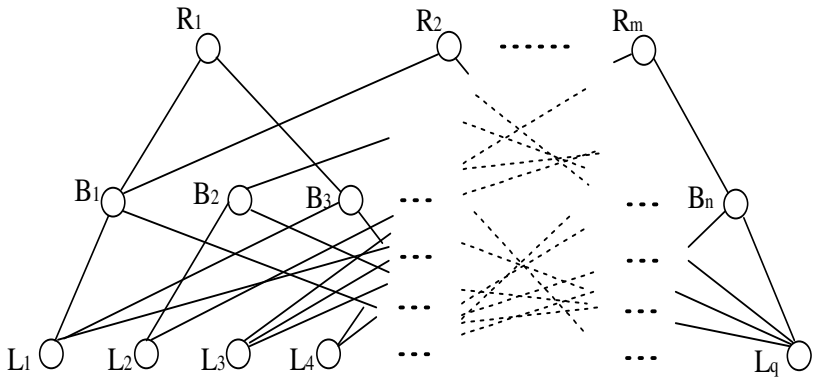

Figure IV-19 : Example of crossing forest

Two key characteristics can be drawn from the figure for the nodes at root level, namely R_1, R_2, …, R_m, each has strictly two children. Different ancestor nodes may share the same descendant nodes. The same is true with all nodes in medium level. Here, we define Crossing Forest as the combination of multiple dimidiate trees with the same nodes at each level merged. The difference between the forest of data structure and ours is the existence of shared descendant. Crossing forest reveals the relationship between nodes in different levels which respectively indicates the given inputs, the outputs of first level adder array and the outputs of the second level adder array which are required by applications**.**

According to the uncertainty property of dimidiate tree, when the nodes in root and leaf levels of crossing tree are known, the nodes that satisfy the requirement of medium level in the forest are not unique. Now, the targeted problem can be described as how to find out the best nodes set in medium level with the minimum number when the nodes in the other two levels are given. From each child node side, they all have as many parents as possible.

IV.5.4. Algorithm Searching for Best Set

Because of the differences in the definition and properties between dimidiate tree and binary tree, existing binary tree theorem and algorithm do not apply to our case. The algorithm for optimal set for medium level is discussed in detail. Before the introduction of algorithm, some properties of nodes in root and leaf levels in crossing forest are defined to make the analysis logical.

The nodes in root and leaf levels are not the arbitrary ones. We have the following definitions based on the architecture introduced in section IV: each node in root level is the set with three or four elements; each node in medium level is the set with exactly two elements and each node in leaf level is the set with exactly one element. The number of elements in each node determines the level the node belongs to. The sets, R_1, R_2, …, $R_{m,}$ of root level are dissimilar to each other and satisfy the equation,

$$R_1 \bigcap R_2 \bigcap, ..., \bigcap R_m = \varnothing \quad \textbf{(IV-1)}$$

where m is the nodes number. We define R as

$$R_1 \bigcup R_2 \bigcup, ..., \bigcup R_m = R \quad \textbf{(IV-2)}$$

Then, the relationship between R and sets, L_1, L_2, …, $L_{q,}$ of root level satisfies

$$L_1, L_2, ..., L_q \subset R \quad \textbf{(IV-3)}$$

where q is the nodes number. We use B_1, B_2, …, $B_{n,}$ to denote the nodes fulfilling the requirement of crossing forest medium level. It is assumed that there is a set of sets, B, satisfying the requirement of optimal set, whose element number is the smallest among all possible sets. In set B, there must exist such subsets B_1, B_2, …, B_p and each subset has the most frequent employment times when it is used for constructing the nodes root level. According to the relationship among the nodes in root, medium and leaf levels, a node in each level can be considered a combination of its down level that is a node in root level could be the combination of two nodes in medium level or the combination of a node in medium level and a node in leaf level; the node in medium level is the combination of exactly two nodes in leaf level. Therefore, even the number of nodes in leaf level is known the numbers of nodes in root and medium levels cannot be determined. Take the number of nodes in medium level, p, as an example. The value of p varies from 1 to C_q^2, where q is the number of nodes in leaf level. Once the sets B_1, B_2, …, $B_{p,}$ are known, the rest sets in B will be obtained by subtracting the relevant set from R_1, R_2, …, R_m.

h \ k	0	1	2	……	q−2	q−1
0	0	$P_{0,1}$	$P_{0,2}$	……	$P_{0,q-2}$	$P_{0,q-1}$
1	$P_{1,0}$	0	$P_{1,2}$	……	$P_{1,q-2}$	$P_{1,q-1}$
2	$P_{3,1}$	$P_{3,2}$	0	……	$P_{2,q-2}$	$P_{2,q-1}$
⋮	⋮	⋮	⋮		⋮	⋮
q−2	$P_{q-1,1}$	$P_{q-1,2}$	$P_{q-2,2}$	……	0	$P_{q-2,q-1}$
q−1	$P_{q,1}$	$P_{q,2}$	$P_{q-1,2}$	……	$P_{q-1,q-2}$	0

Figure IV-20: Coefficient ($P_{h,k}$) matrix

Now, we simplify the targeted problem as finding out the new set containing subsets $B_{best\_all}=\{B_1, B_2, \ldots, B_p\}$. A key property of the new set can be concluded as follows: each element in this set will appear as frequently as possible in the nodes of root level. On the other hand, B_{best_all} has the least elements among all possible sets which satisfy the requirements of dimidiate tree. Based on this property, we construct new subsets of $\{R_1, R_2, \ldots, R_m\}$, denoted with R'_h, where $h \in (0, 1, 2, \ldots\ldots, q-1)$. It is the set that contains all the sets in the root level with L_h element. Obviously, the number of new sets is equal to the number of nodes in leaf level. We can easily find out the element which appears most frequently in R'_h:

We define $P_{h,k}$ as representing leaf L_h ($h \in \{0, 1, 2, \ldots\ldots, q-1\}$) appearance time in set R'_h. A matrix of $P_{h,k}$ can be obtained from $\{R'_1, R'_2, \ldots, R'_h\}$, as shown in Figure IV-20. If $P_{h,max}$ is the symbol for the largest one in $\{P_{h,1}, P_{h,2}, \ldots\ldots, P_{h,k}\}$, the pair $\{L_h, L_k \mid P_{h,k} = P_{h,max}\}$ will be the best pair which is employed the most in set R'_h. This pair is also one element of set $B_{best\_all}=\{B_0, B_1, \ldots, B_{p-1}\}$. Following this method, all pairs for $R'_1, R'_2, \ldots, R'_h$, can be obtained. It is noted that the duplication pair must exist among them, which is caused by the uniqueness and randomness of pair $\{L_h, L_k\}$. All pairs are generated with the index of L_h and L_k, in which h and k vary from 0 to $q-1$. After elimination of duplications, the set $B_{best\_stage1}=\{B_{st1\_0}, B_{st1\_1}, \ldots, B_{st1\_p-1}\}$ is determined. But it is just a subset of set B_{best_all}.

According to the property of dimidiate tree, each node in root level has strictly two children which may be shared with other root nodes. The rest child nodes of one root node must be one element of B_{best_all} when the corresponding pair $\{L_h, L_k\}$ is taken from its parent node. As a result, we have another set $B_{best\_stage2}=\{B_{st2\_0}, B_{st2\_1}, \ldots, B_{st2\_p-1}\}$ whose elements must be included in the set B_{best_all}.

The same method is adopted after the finished root sets are taken from B_{best_all} until all nodes in root level have their children nodes. The application of the algorithm to DCT will be detailed in the next chapter.

IV.5.5. Software Implementation

In order to apply the algorithms for obtaining optimal common terms sharing scheme automatically, a program is implemented based on the algorithms introduced and discussed in previous sub-sections. This program is available for any DA applications, which can generate the coefficient matrix automatically for given application and output the best common terms sharing scheme according to our architecture.

To make the introduction and discussion of program easy to understand, let's begin with the definition of DA. Equation (III-3) can be re-written in the form of matrix product as shown below

$$Z = AX = \sum_{i=0}^{L-1} C_i X_i$$

$$= [2^0 \quad 2^1 \quad \cdots \quad 2^{M-1}] \begin{bmatrix} C_{0,0} & C_{1,0} & \cdots & C_{L-1,0} \\ C_{0,1} & C_{1,1} & \cdots & C_{L-1,1} \\ \vdots & \vdots & \cdots & \vdots \\ C_{0,M-1} & C_{1,M-1} & \cdots & C_{L-1,M-1} \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{L-1} \end{bmatrix} \qquad \textbf{(IV-4)}$$

where $C_{i,n}$ is the nth bit of the constant C_i, and $C_{i,n} = 0$ or 1, $C_{i,0}$ and $C_{i,M-1}$ are the LSB and MSB of C_i respectively. The real constant coefficients are represented with M precision in 2's complement bit representation. When the 2's complement bit matrix of coefficients is obtained, the duplicated lines in the matrix are removed to make each line unique. Then, a reduced matrix is generated in which each line is different from others. In the following step, the reduced coefficient matrix with representation of 0 and 1 is converted to the set of part

product in format of X_i. Taking the line [0 0 1 1 0 0 1 1] as an example, it is converted to the set, $(X_2X_3X_6X_7)$, in which the position of 1 is replaced by the corresponding input vectors.

This set is one of the root nodes of the dimidiate tree and will build up set R_S = { R_1, R_2, …, R_M }, along with other sets. For a single output Z, there is up to M lines in the coefficient matrix if there is no duplicate one among them. It means the maximum elements number of set R'_h is M for a signal output mode. The set R_S represents a single output, as indicated by the subscript 'S'.

For the multi-output architecture, just like the processor proposed in this dissertation, the set for total root nodes is denoted with R, which is defined as

$$R = Rs_0 \bigcup Rs_1 \bigcup Rs_2 \bigcup, ..., \bigcup Rs_Q \quad \textbf{(IV-5)}$$

where R_{S0} is the root nodes set for output Z_0. For our architecture, the maximum output number is 8 then the root nodes set of the dimidiate tree of proposed architecture is

$$R = Rs_0 \bigcup Rs_1 \bigcup Rs_2 \bigcup, ..., \bigcup Rs_7 \quad \textbf{(IV-6)}$$

It is noted that the elements in the set R are unique and the reduplicate elements among all subsets $\{R_{SQ}\}$, $Q \in \{0,1,\ldots\ldots 7\}$ have been removed. By now the set R has been specified. As described in the section IV.5.4, the rest step is to set up R'_h, a subset of R which contains all the sets in the root level with L_h element, and matrix of $P_{h,k}$, a matrix containing leaf appearance time L_h, based on the known R.

The algorithm and ideas for searching for optimal common terms sharing is implemented with Matlab 7.0 because of its powerful built-in mathematical functions and extensive application-specific function libraries.

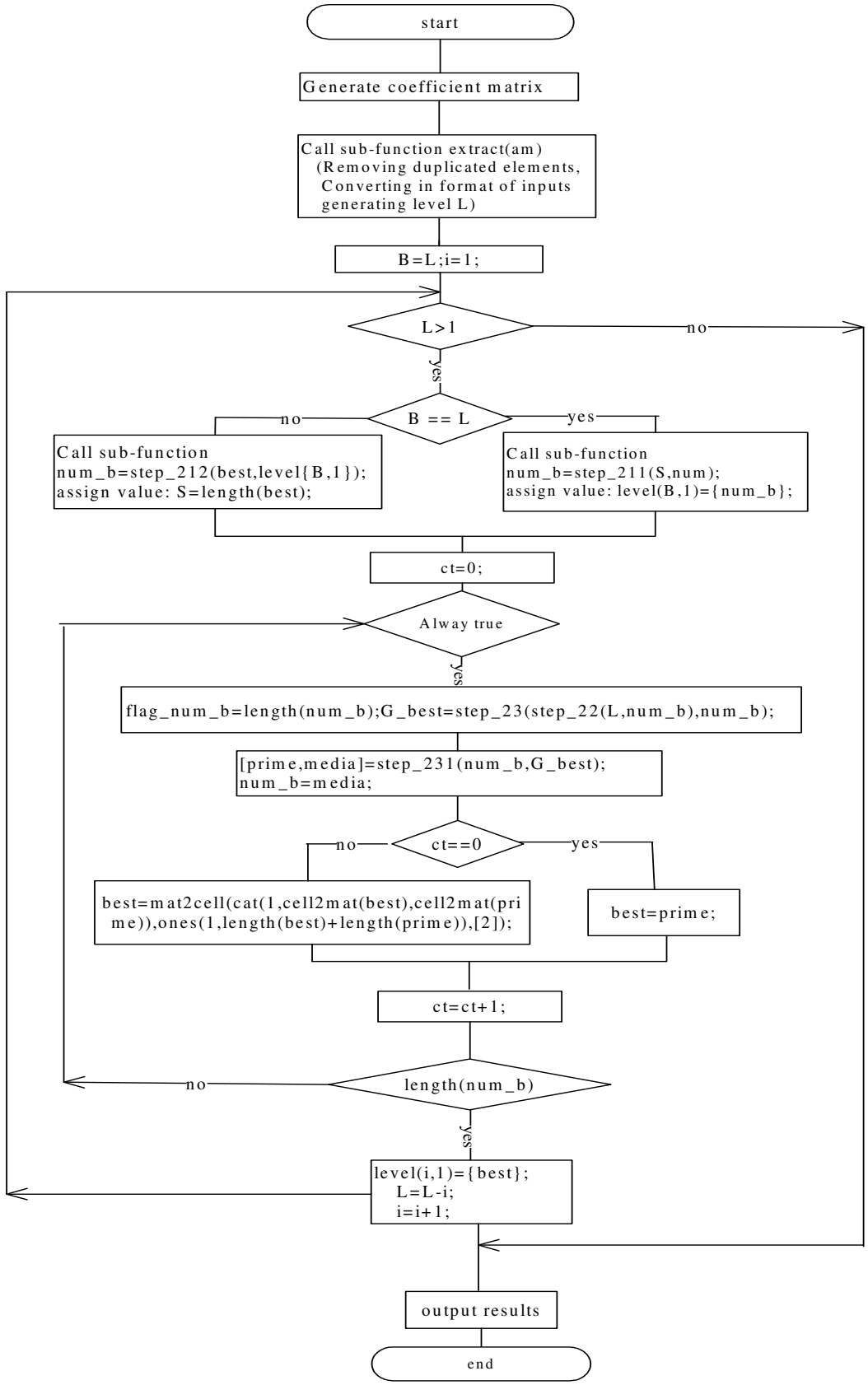

Figure IV-21 : Design flow chart

This program is not only applicable to the proposed architecture in which two-level adder matrix are adopted. Since the developed algorithms are general, the program for implementing these algorithms can be directly applied to any application in which the dimidiate tree characteristics are satisfied. Therefore, the number of levels is generated for controlling the performing times of sub-function. This level number can be obtained with Equation (IV-7)

$$L = \lceil \log_2 N_{MAX} \rceil \qquad \textbf{(IV-7)}$$

where N_{MAX} is the maximum number of "1"s in each line of coefficient matrixes. $\lceil\ \rceil$ is ceiling symbol, which truncates a number toward positive infinity and returns the smallest integer not smaller than input. Taking the line [0 0 1 1 0 0 1 1] in $P_{h,k}$ coefficient matrixes as an example, it is converted to the set, ($X_2X_3X_6X_7$). This means that two-level dimidiate tree is required for implementation. For the case ($X_2X_3X_6\ X_7X_8X_{10}X_{11}X_{12}$) which has eight elements, a three-level dimidiate tree is necessary.

When the duplications in set R are removed in sub-function *step_211*, the coefficient $P_{h,k}$ matrix for leaf L_h ($h \in \{1, 2, \ldots, q\}$) appearance times in set R'_h can be obtained in sub-function *step_22* when h ranges from 1 to q. The set B_{best_stage1} can be specified in sub-function *step_23,* in which the pair of $\{L_h, L_k\}$ with maximum $P_{h,k}$ in each line of matrix is selected. After that, the set B_{best_stage2} can be easily obtained in sub-function *step_231.*In this sub-function, the program will check whether the selected sets can match all the nodes in root level.

If unmatched root nodes still exist when above steps are done, they will be sent back to sub-function *step_212* for further processing. By now, the rest root nodes construct a new set which is a subset of set R. To treat this new set, just follow the same processing steps on set R until no unmatched root nodes exist anymore. Then, we can get the best common terms sharing scheme in the format of the pair with two single elements. As we have discussed in previous sub-sections, the nodes in root and leaf levels are known. The purpose of the algorithm is to find out the best set for medium level. For the cases which require two-level dimidiate tree, the best sets we have are the final result, which build up the only medium level.

For the cases which require a dimidiate tree with more than two levels, there is more than one medium level. The obtained sets can only construct the medium level which is most close to leaf level. We still need to find out the other medium levels. For these cases, each set we got will be re-coded with a single symbol and build a new set in sub-function *step_212*. Just following the same processing steps for set *R* to process this new set, we will get some new sets. These sets will set up another medium level which is the one upper than the medium level we already have. Taking the case with the one root node ($X_2X_3X_6$ $X_7X_8X_{10}X_{11}X_{12}$) as an example, suppose sets (X_2X_3), (X_6X_7), (X_8X_{10}) and ($X_{11}X_{12}$) are ones of the best pairs. We use a single symbol to indicate each pair. There are A_1-> (X_2X_3), A_2-> (X_6X_7), A_3-> (X_8X_{10}) and A_4-> ($X_{11}X_{12}$). The original root node ($X_2X_3X_6$ $X_7X_8X_{10}X_{11}X_{12}$) then will be re-coded as ($A_1A_2A_3A_4$). After all the original nodes are re-coded in the format of *A*, another round of processing for searching for best common terms sharing scheme will be performed as described previously until all are done

IV.6 Comparison with Subexpression Sharing in CSD

The idea behind common term sharing in two's complement binary multiplier is not new, which can be found in [82-86], namely common subexpression sharing or common subexpression elimination. The purpose of common term sharing is the same as common subexpression elimination's, which is to reduce the number of adders. However, the implementation strategies for two methods are greatly different.

Common subexpression elimination is to find multiple common subexpressions in the coefficient set. The efficiency of common subexpression elimination is based on canonic-signed-digit (CSD) code which is widely used as signed-powers-of-two code for its minimal number of nonzero digits.

The CSD number system is a signed digit number system that minimizes the number of non-zero digits and thus can reduce the number of partial product additions in a hardware multiplier. The encoding scheme uses a digit set that is ternary and each digit can be either -1, 0, or +l. The signed-digit representation is named canonical if it contains no adjacent nonzero. For a 2's complement number, there can be n non-zero digits for an n-bit number. It

can also be shown that the probability of a digit being zero is roughly 2/3 for CSD and exactly 1/2 for 2's complement [87, 88]. For example, we compute the canonical recoding of $x = 478 = (0111011110)_2$ by starting with $c_0 = 0$, and then compute y_i and c_i+1 using x_i+1, x_i, and c_i with $_i$ = 0, 1,……, 9. The resulting vector is y = 1000100010. Encoding the coefficients of applications using the CSD representation reduces the number of partial products and thus saves silicon area and power consumption in hardware implementation. It is well known that the CSD representation can be used to reduce the complexity of applications implementation such as DCT [89], DFT [86, 90],FIR digital filter [91-93] and so on. But the high efficient code is obtained at the cost of extra circuit because of twice conversions between 2's complement number and CSD. It will consume more silicon area and power in hardware implementation and then introduce extra delay time to system.

The idea common term sharing and common subexpression sharing is similar, but the implementation for them are different regardless of the extra processing of coding for CSD. Taking an example to show how common subexpression sharing works, for *Y* which is

$$Y = c \cdot X \qquad \textbf{(IV-8)}$$

where *X* is input. Assuming $c = 0.6458 = 0.101001010101 = 2^{-1}+ 2^{-3}+ 2^{-6}+ 2^{-8}+ 2^{-10}+ 2^{-12}$, the output *Y* can be expressed as

$$Y = X \gg 1+X \gg 3+X \gg 6+X \gg 8+X \gg 10+X \gg 12 \qquad \textbf{(IV-9)}$$

where the sign "» n " indicates a n-step right shift operation. In this case, five intra-structure adders are required to obtain *Y*. By applying common subexpression sharing, Equation (IV-9) can be rewritten as

$$Y = u \gg 1+ u \gg 6+ u \gg 10 \qquad \textbf{(IV-10)}$$

where u is defined as

$$u = X+X \gg 2 \qquad \textbf{(IV-11)}$$

Hence the multiplication structure optimized using the common subexpression sharing given by Equation (IV-10) and Equation (IV-11) requires two adders which is less than the original structure of Equation (IV-9). Thus using common subexpression sharing, the number of adders required to implement the multiplication is minimized. According to the characteristics of common subexpression sharing, the operation is a serial one, from right to left. The common subexpression sharing occurs inside one coefficient. Compared with these

characteristics, common term sharing in proposed architecture is a parallel operation which will improve the system performance in terms of speed**.** On the other hand, the shared common terms are located at the global level, not limited to one coefficient. The common terms are based on the searching of the entire coefficients for the application. For example, the shared common terms for 8-point 1-D DCT with 12-bit precision in coefficient are based on the searching result for total 96 (=8x12) coefficients. This characteristic makes our architecture take full advantage of terms sharing and greatly improves the hardware efficiency.

In addition, common term sharing does not make any changes in 2's complement code and algorithm expression. The comparison between two methods targeting at the same application will be made in the next chapter.

IV.7 Conclusion

This chapter has presented low power reconfigurable DA architecture and its implementation. The research work carried out on DA techniques and their implementations in DCT and other applications, reconfigurable architectures and their implementations and a domain-specific RA targeting DA was first introduced. The overview architecture was described and then the detailed descriptions were given for the algorithm logic unit. The control unit of proposed processor is separated and will be introduced and described in Chapter V because of its complexity and uniquely important function. In Chapter V, the performance on area, power and delay of control unit will be evaluated and analyzed as well.

The algorithm logic unit has been described in detail in the introduction of each function module inside including two-level adder structure, Wallace tree multiplier matrix, interconnection network and memory unit for reconfigure bits. According to the two-level adder array adopted in our architecture, common terms sharing has been introduced in this chapter including its availability analysis, optimal scheme searching algorithm and matlab implementation and comparison with subexpression sharing in CSD. To obtain optimal scheme searching algorithm, the concepts of dimidiate tree and crossing forest have been

introduced and defined and accordingly the algorithm has been developed to get the architecture mapped with the best efficiency.

Chapter V

Reconfigurable Control Unit

V.1. Overview

In digital systems design the use of control units to implement complex system behaviour is ubiquitous and found in almost all engineering disciplines. Conventional controllers can be micro-coded or hardwired finite state machines (FSMs). They make the systems which they integrated have the abilities to allow devices to operate in an intelligent manner rather than simply reacting to the current operating conditions of the system.

The controller design is the implementation of application-specific algorithms**.** In some cases, software implements the bulk of algorithms, because microprocessors are flexible and easily programmable for a wide range of functions with dedicated hardware to acquire and store data on behalf of the software. The use of microprocessors allows a device to freely determine its response to a sequence of events according to different algorithms rather than simply producing a response based on single event. Some systems may not be able to perform their intended tasks with software alone and the reasons for this vary by applications and often include throughput problem that systems with a microprocessor do not practically meet some requirements of real-time applications.

State machines are often adopted to accomplish the load when control task is implemented in hardware. Hardwired FSMs implement the controller output functions and next state logic using fixed logic gates, and are the more common choice given their relatively small area and higher performance. A state machine can be made arbitrarily complex and can function similarly to software running on a microprocessor. Just as software moves through a sequence of tiny steps to solve a larger problem, a state machine can be designed to advance

when certain conditions are satisfied. As the state machine progresses, it can activate other functions, just as software requests transactions from a microprocessor's peripherals.

Although it is often unnecessary to explicitly specify the control system for simple applications, it is inferred from the functional description of the application. For more complex applications a high-level model is required to allow designers to specify the control behaviour of the system. A commonly used high-level model is the FSM. FSMs are abstract models used to describe the behaviour of a control system in response to a sequence of inputs and are based on the theory of finite automata in computer science [94].

To implement FSMs, two steps are generally followed in the procedure: the extraction of the FSM from the high level specification of the system and the implementation of the FSM. The extraction of FSMs is independent of the method of implementation and is concerned with developing a high-level description of the control aspects of the specification. The implementation in dedicated hardware involves the production of an ASIC that implements the required behaviour. This approach produces devices with low area and power requirements but requires lengthy and expensive hardware design and manufacturing processes, making it impractical for low volume applications or for developers with limited resources.

As FSMs are used in almost all control applications, the use of a reconfigurable hardware device specifically tailored for the implementation of these control systems has a wide market. Using reconfigurable devices such as FPGA devices is an important hardware implementation of FSMs. Reconfigurable hardware combines the speed and efficiency of hardware with the flexibility and programmability of software. This flexibility and programmability requires the introduction of programmable hardware devices which result in a lower efficiency when compared to full-custom ASIC design.

Domain-specific reconfigurable technology has emerged to cover this gap, which has the ability to be programmed according to the requirements of target application. But, due to the unique flexibility of FSMs, it is difficult to implement it with a reconfigurable architecture with reasonable resource. Finite state machines are always too complex to map to a

reasonable number of very fine-grained logic blocks [14]. However, finite state machines are also too dependent upon single bit values to be efficiently implemented in a very coarse-grained architecture. This type of circuit is more suited to an architecture that provides more connections and computational power per logic block, yet it still provides sufficient capability for bit-level manipulation. A typical very fine-grained architecture is composed of logic block which is the configurable functional unit embedded in a symmetry interconnection network sea and can implement any two-input function and some three-input functions by loading different reconfigure bits. However, although this type of architecture is useful for very fine-grained bit manipulation, it can be too fine-grained to efficiently implement many types of circuits, such as multipliers [14].

This chapter of the portfolio is intended to give details of the work carried out on the development of reconfigurable hardware for FSMs implementation. This method makes use of a novel architecture that allows the implementation of these machines to consume far less resources than commercial devices and traditional architecture. This chapter begins by presenting the formal definition of FSMs, categories, traditional representation and decomposition of FSMs in section V.2 which is followed by an overview of the traditional design flow of implementation for a range of FSMs in section V.3. Existing FSM implementation in commercial PLD and CPLD devices is first introduced briefly and then detailed by analysing Xilinx Coolrunner CPLD in terms of their benefits and limitations in section V.4. Existing reconfigurable customer-specific hardware for FSMs is discussed in this section as well for comparison with the novel method presented here.

The novel reconfigurable architecture of FSMs suggested is then presented in section V.5 including the overview of the architecture, functional sections of the architecture, logic & sequential block and the construction of PTB inside the block. Section V.6 then presents the low power implementation in interconnection network, PTB construction and mapping of PTB. In order to prove the usefulness of the proposed architecture it is then compared to the traditional implementation of FSMs and the results of this comparison are presented in section V.7 before final conclusions are made in section V.8.

V.2. Background

In order to make the novel method in the implementation of the proposed architecture easy to understand, it is necessary to first understand both the theory and implementation of FSM. This section begins by presenting a formal definition of FSM which is followed by a description of the process of FSM extraction before the implementation. Hardware FSMs implementation is discussed for both the ASIC and reconfigurable device design flows.

V. 2. 1. Definition of FSM

In digital systems, control logic can be expressed as a sequence of states and state transitions. A behavioural specification can be implemented with an FSM. FSMs are abstract models used to represent the sequential behaviour of systems. They are used in control applications to define the response of a system to a sequence of input events. This allows designers to implement systems with complex behaviour [94].

A Finite State Machine is a 6-tuple (I, O, S, R, T, S0):

- I is a finite set of inputs of the FSM.
- O is a finite set of outputs of the FSM.
- S is a finite set of internal states of the FSM. It consists of two parts: Sc and Sn. Two symbols present current states and next states respectively.
- R (i, s) is a relation from the (input, current states) pairs to next state (i.e., R I×Sc×Sn).
- T (i, s) is a relation from the (input, current states) pairs to the outputs (i.e., T I×Sc×O).
- S0 is an initial (or reset) state.

An FSM is deterministic, if for each pair (i, p)∈I×S, there exists at most one next state ‘n’ and one output ‘o’ such that (i, p, n, o)∈R×T(It means they are defined for all elements of their domains), otherwise FSM is nondeterministic. If at least one transition is specified for each present state and each input, an FSM is said to be completed; otherwise, the FSM is partial. We will call an incompletely specified deterministic FSM simply an FSM. State transitions are assigned to each state and based on the current inputs and current state are

used to determine the next state of the FSM. The current state of the FSM is determined by the initial state of the FSM and the previous input sequence applied to the machine. This means if the previous input sequence and start state of the FSM is known, it is possible to predict the current state of the FSM.

V. 2. 2. Three Categories of FSM

From the output function point of view, there are three cases found in the practical implementations. If the outputs of an FSM are always associated with the inputs, the FSM is a Mealy machine, shown in Figure V-1.

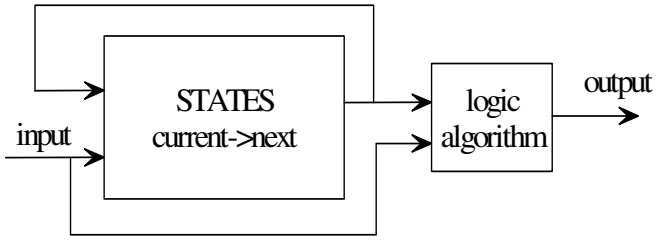

Figure V-1: Mealy machine

If the outputs are influenced directly by the internal states of the FSM and inputs affect the outputs only through the states, the FSM is a Moore machine, see Figure V-2.

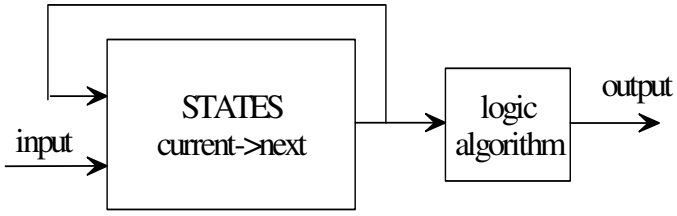

Figure V-2: Moore machine

For the third case, the internal states of the FSM are output straightway as the result of FSM, shown in Figure V-3.

Figure V-3: FSM with internal states as outputs

V. 2. 3. State Transition Graph and State Transition Table Representation of FSM

A directed graph can also describe an FSM, called the state transition graph. In a state transition graph, each vertices denotes an internal state, and each edge corresponding to the state transition is labelled with an input /output pair, and is directed from current state vertices to next state vertices.

A counter is a simple example of an FSM, though its actions are very limited. Each state simply advances to the next state on each clock cycle. There is no conditional branching in a typical counter. FSMs are often represented graphically before being committed to Register Transfer Level (RTL). Figure V-4 shows a diagram representation of a two-bit counter. Each state is presented by its own bubble, and architectures show the conditions that cause one state to lead to any other state**.** An unlabeled architecture is taken to mean the default if no other condition is valid. Because this is a simple counter, each state has one unconditional arc that leads to the next state.

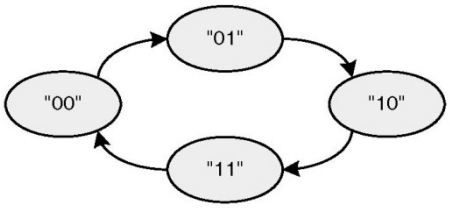

Figure V-4: State transition graph for two-bit counter

An alternative to the state transition graph (STG) is the State Transition Table (STT). A state transition table is a truth table that shows the output and next state for a synchronous sequential network, for a given input and current state. The table shows each of the required state transitions of the FSM and gives the current state and input conditions required to cause these transitions. The STT of the Mead-Conway traffic light controller is shown in Table V-1.

Table V-1: State transition Table of the case

Current state	Inputs	Next state
Idle	==0x01	Wait02
Idle	!=0x01	Wait01
Wait01	==0x01	Wait02
Wait01	!=0x01	Wait01
Wait02	==0x01	Wait02
Wait02	==0x02	Wait03
Wait02	!=0x01 && !=0x02	Wait01
Wait03	==0x01	Wait02
Wait03	!=0x01 && !=0x03	Wait01
Wait03	==0x03	Wait04
Wait04	==0x01	Wait02
Wait04	!=0x01 && !=0x04	Wait01
Wait04	==0x04 / Match=1	Idle

V. 2. 4. Decomposition of FSM

It is often convenient to realize a sequential circuit as an interconnection of two or more sub-circuits. The decomposition may be useful for both area and performance reasons. The decomposition of FSMs is adopted to improve the hardware efficiency in the complex cases.

The proposed reconfigurable FSM architecture is a hierarchical system which can implement generic FSM for control system. The capability of the architecture is so large as to implement a generic FSM with a maximum of 256 (2^8) states with up to 8 inputs and can be extended by increasing the number of sequential and logic blocks. FSM decomposition is employed to overcome the limit on the size for the case of large FSM. The decomposition of FSM has been proved power-saving [95-97]. In theory, any large FSM can be decomposed into a desired set of smaller sub-FSMs. Therefore, any size FSM can be implemented with our architecture with a maximum number of 256 states after decomposition.

The initial FSM decomposition work dated back to 1960 by Hartmanis [98] and has been further developed by several researchers [99, 100]. The structure of the resulting finite state machine is shown in Figure V-5. The inputs of each sub-machine are not only the primary inputs and its own last state, but also the state variables from the sub-machine after decomposition.

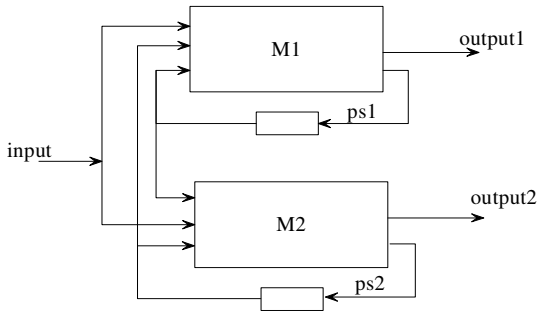

Figure V-5: Generic FSM decomposition

To illustrate the procedure of FSM's decomposition, consider the state transition graph of a simple FSM shown in Figure V-6.

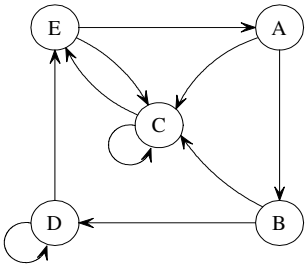

Figure V-6: State transition graph of an example FSM

It is assumed that a desired decomposition of the corresponding FSM is given that state A and state B belong to sub-machine T2 and all the other states belong to sub-machine T1. So the original FSM is divided into two smaller sub-FSMs which are depicted with two STGs as shown in Figure V-7.

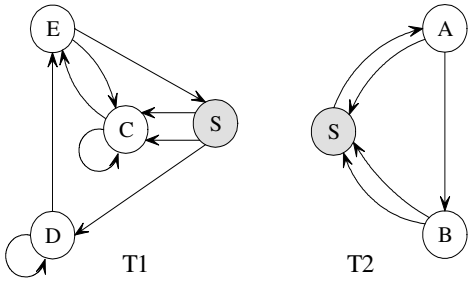

Figure V-7: State transition graph after decomposition

As a result of the decomposition, a RESET state is added to both sub-FSMs, labelled with S. The transitions between the states except the state S in T1 and T2 are the same as the original FSM without transformation. As shown in Figure V-7, the number of inputs to each sub-FSM is also changed because of the additional RESET state. In each sub-FSM, the extra inputs are required for the purpose of communication between two sub-FSMs, so do for the extra outputs.

Following this approach, a large original FSM can be decomposed into several smaller ones with limited states which fit our architecture well and can be implemented easily.

V.3. Implementation of FSMs

The implementation of FSMs can be generally split into two sections, the extraction of the FSM description from specification of the application and the implementation of the FSM with the appropriate hardware platform. The FSM design flow for both full-custom ASIC and reconfigurable hardware devices is shown in Figure V-8. It can be concluded from the figure that the extraction of the FSM is independent of the implementation technology and is hence applied in both design flows.

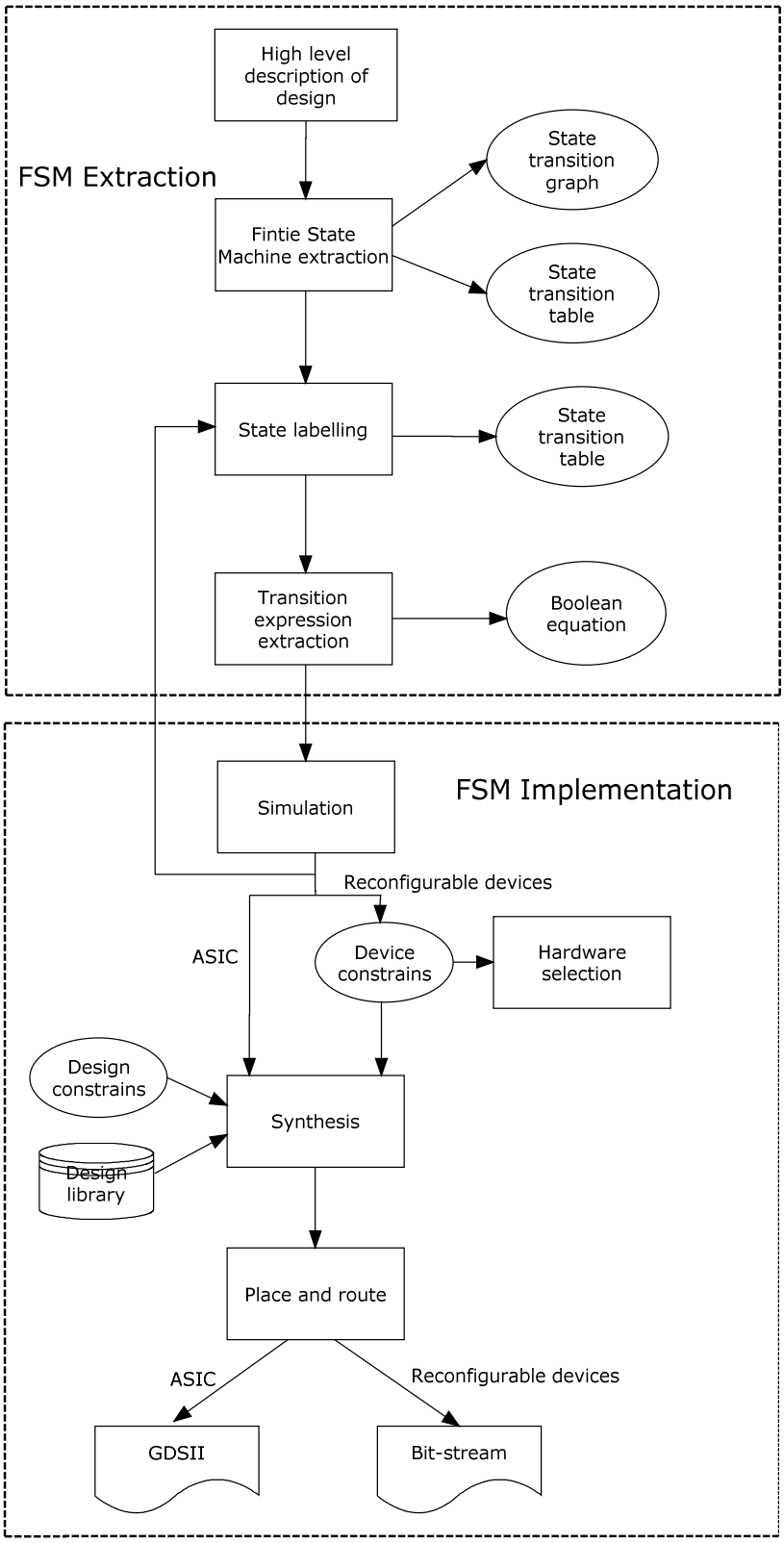

Figure V-8: FSM design flow

V.3.1. Extraction of FSM Implement FSM with Hardware Platforms

In Figure V-8, the conventional method of FSM extraction is shown. A high level description of the control requirements of the proposed device are first extracted from the initial specification of the system. This behaviour can then be expressed as an FSM using a high level model such as a STG or STT. Based on this model it is then possible to produce a description of the FSM suitable for the implementation of the control requirements.

After extraction of FSM, the Boolean description of FSM is generated. The main step for hardware implementation is commonly known as synthesis which involves taking a high level description and converting this to a description that can be used to produce hardware implementation including ASIC and reconfigurable device. The conventional method of implementing an FSM on both ASIC with standard cells and a reconfigurable hardware platform is shown in Figure V-9.

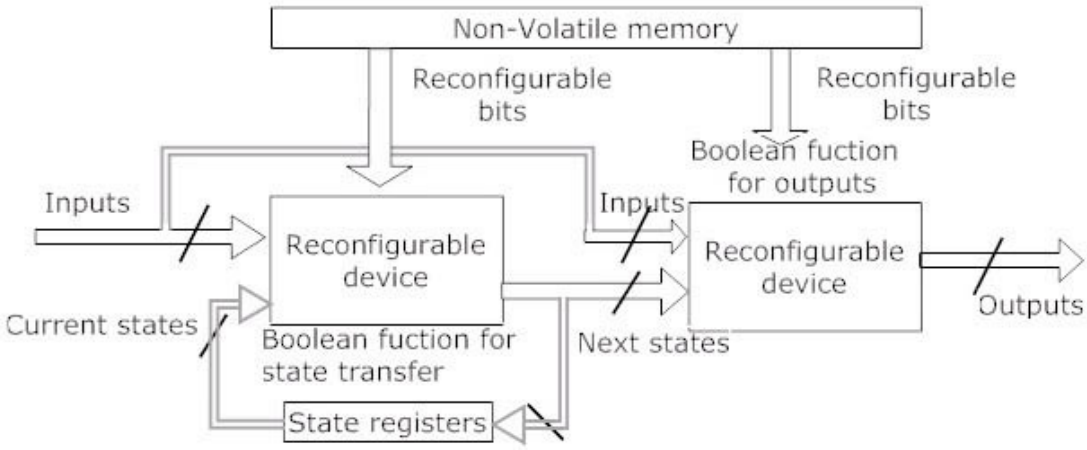

Figure V-9: General reconfigurable hardware implementation of FSMs

The implementation of a FSM in full-custom hardware involves producing a description of the required functionality in a hardware description language (HDL) such as Verilog HDL or VHDL. This description in a high level programming language is capable of producing a silicon implementation of the device.

The process flow for implementation of FSMs with different reconfigurable devices is similar to the design flow used in full-custom ASIC design, but, in full-custom design flow synthesis results in a description that can be used to produce a silicon implementation of the

FSM, and synthesis for reconfigurable devices results in a bit-stream capable of configuring the device to implement the FSM. The synthesis process can be considered the same for full-custom ASIC and different reconfigurable hardware platforms. The difference exists in the result of synthesis which can be obtained in format of netlist with standard cells in ASIC design and is generated in certain format in reconfigurable hardware implementation but invisible to designer.

V.3.2. FSM Operation on Reconfigurable Device

The conventional hardware FSM implementing model is shown in Figure V-9. For reconfigurable hardware platform, the reconfigurable function units will be programmed to realize the required function which is expressed in Boolean equation. Take Xilinx Virtex-E device as an example [10], it comprises two major configurable elements: configurable logic blocks (CLBs) and input/output blocks (IOBs). CLBs provide the functional elements for constructing logic while IOB**s** provide the interface between the package pins and the CLBs. A general routing matrix (GRM) makes the connection between CLBs, which comprises an array of routing switches located at the intersections of horizontal and vertical routing channels. The architecture of CLB of Virtex-E is shown in Figure V-10.

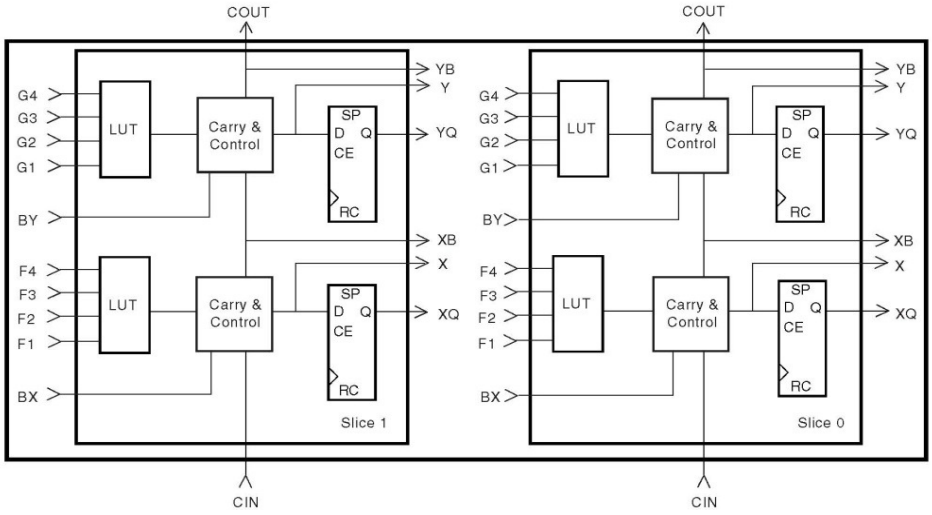

Figure V-10: CLB of Xilinx Virtex-E FPGA [10]

In FSM implementation with FPGA devices, CLB realize desired Boolean functions to generate next states and output. Current state register used to store the current state of the FSM is implemented with the flip-flop within CLB. When it is powered on, the

configuration bit-file is loaded into the local memory within the reconfigurable device from non-volatile memory. This configures the reconfigurable device to implement the required transition expressions and the current state register. When configuration bit-file is loaded onto the device the output would be calculated using the current state and input. After the loading of reconfiguration bits, FSM would be initialized first, the process of which would involve a specified reset state and the current inputs. At the same time, the next state would be calculated using the current state as well. The system would then be clocked and the next state is stored as the current state in the registers. This new current state would then be used to calculate the next state again and output before the device is clocked. Because the calculation of output is a combinational logic and generally without clocked data buffer, the clock speed of the device is limited by the time required for the device to calculate the next state and store it as the current state.

V.3.3. Reconfigurable Hardware Platform for FSMs Implementation

The main stream commercial reconfigurable devices are FPGA and CPLD/PLD as described in previous section. Different devices provide a wild range of benefit, capability and performance in area, power and delay. The selection of a suitable reconfigurable device is the key point to the efficient implementation of the FSM. It is essential to ensure that the device has sufficient hardware resources to implement the FSM without introducing excessive redundant hardware. Being a pre-design part, the integrated reconfigurable part is fixed for its size, area and capability. It is impossible that the device has exactly the right amount of hardware required to implement a particular application. However, devices are available from device manufacturers in a wide range of sizes and costs, which makes the designer to select the device most close to the desired application.

For the FSMs implementation, the high level description of the FSM makes designers to extract an estimation of the hardware requirements of the device. The end-user would then select the device with as close to these parameters as possible to ensure that excessive hardware is not introduced that would have impact on the cost, area and possibly power of the final implementation.

V.4. Existing FSM Hardware Implementation Architectures

The existing FSM hardware implementation architectures can be separated into two main groups, the commercial FPGA/CPLD devices and customer-specific reconfigurable architecture. The FSMs implementation flow with commercial FPGA/CPLD devices is discussed in previous sub-section. The conventional architecture of FPGA is introduced in detail in section II.2.1. In this section, the architecture of PLD/CPLD will be given in detail. The analysis of its specification and limitation will be followed. In order to familiarize the reader with typical device specifications, we also provide a close examination of a popular commercial CPLD, the Xilinx CoolRunner XPLA3 (eXtended Programmable Logic Array). At the rest part of this section, the existing customer-specific reconfigurable architecture will be introduced and analysed.

V.4.1. PLD Hardware Platform for FSMs Implementation

PLDs devices have the capability to be dynamically reconfigured while implementing the target application by adjusting the memory contents which control routing and logic resources. Then, the routing structure and the functions are implemented with PLDs devices. Several kinds of memory are adopted in the creation of PLDs: EEPROM, flash memory and SRAM. Because EEPROM and flash have the characteristic of keeping their contents without power, they are employed in most PLDs devices.

The basic method to perform logic in PLDs is to use PLA, a programmable logic array. PLAs is such an architecture that directly implements two level sum-of-products Boolean functions. It is implemented with a programmable AND-plane which is followed by a programmable OR-plane. The input signals arrive at the array in both true and negated forms simultaneously, not all but only the appropriate signals are fed to the AND gates as inputs which are selected by configurable switches. The outputs of the AND-plane are then fed to the gates in OR-plane. The outputs of these OR gates can be used as combinational signals, being determined by the output multiplexer. The actual hardware implementation of a PLA is constructed with NOR-NOR-plane rather than AND and OR-planes, with the former being more efficient than the latter. The idea behind this is that any equation in sum-of-products

form, which is implemented by AND-OR-plane, can be represented easily in NOR-NOR format by applying De Morgan's law.

Another type of array, called Programmable Array Logic (PAL), can also directly perform sum-of-products style Boolean function. A PAL construction is different from a PLA in only one way: a fixed OR-plane replacing the fully programmable OR-plane in PLA. This characteristic makes PALs slightly smaller than PLAs at the cost of less flexibility due to the fixed nature of their OR array.

V.4.2. CPLD Hardware Platform and Xilinx CoolRunner XPLA3 CPLD

The PLDs introduced in last sub-section can be combined to build a popular kind of PLD, the Complex Programmable Logic Device (CPLD). Either PLAs or PALs are employed in CPLDs as their functional units, and are connected together by crossbars in general. Because the size of crossbars will grow exponentially for larger application, CPLDs are historically limited to small and medium sized designs.

The interconnection network in a CPLD is typically a full network in which each input and PLA/PAL output drives a wire separately through full crossbars with the capability to deliver the signals to any port desired. Xilinx CoolRunner XPLA3 CPLD is taken as an example to illustrate the CPLD architecture.

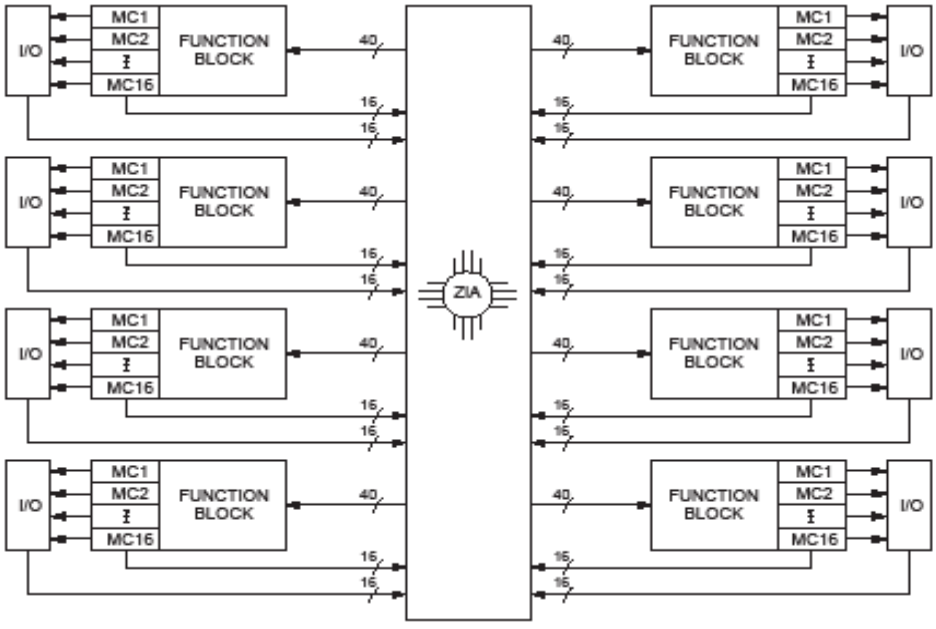

Figure V-11: Xilinx XPLA3 CPLD architecture [101]

The architecture of Xilinx CoolRunner XPLA3 CPLD is shown in [101]. The CoolRunner XPLA3 architecture consists of function blocks, 16 Macrocells and a Zero-power Interconnect Array (ZIA). A function block and 16 Macrocells combine to form a PLA and ZIA acts as a virtual crosspoint switch providing full connectivity between the PLAs. All I/Os and PLA outputs are sent to ZIA and totally 40 PLA inputs are obtained from the ZIA as well. In a PLA, all inputs are fed to the Product-Term Array which can create 48 product terms. There are 16 different outputs generated in function block and each output goes to each corresponding Macrocell. Besides, Xilinx also provides extra functionality to this basic PLA. For example, wider logic equations can be obtained by using eight foldback NAND product terms in each function block. CoolRunner XPLA3 CPLD also provides eight other product terms for controlling the registers in the Macrocell, 16 product terms fed directly to the Macrocell for timing critical signals and Variable Function Multiplexers (VFM) for implementing some two input logic functions. [101]

The Xilinx CoolRunner XPLA3 CPLD provides all of the basic CPLD functionalities and also introduces more hardware which can increase logic density and allows for high-speed signal paths.

V.4.3. Limitation of CPLD

It can be drawn easily from the analysis of CPLD architecture that the method of exhaustion is adopted to implement logic function. All the input and feed-back signals are ready for each product term of a large number of terms. The scale of the matrix used to rout the signals will be great and the area efficiency will be greatly reduced. The method of exhaustion is also incarnated in the construction of AND-OR array. Only one level AND-OR is adopted in the implementation. It means that all the Boolean equation should convert to one level sum of product-term style. It is easy to do, but it results in a low efficiency in implementation.

Obviously, the congenital weakness existing in both FPGA and CPLD architecture leads to the high power consumption and low area efficiency.

V.4.4.Existing Customer-specific Reconfigurable Architecture

In [102] and [103], LUT based and product-term-based reconfigurable architectures are introduced to implement combinational circuit based on a rectangular and triangular constructions respectively. Combinational circuit can be used to implement any Boolean functions if its size has not limitation. Therefore, both architectures can be used in FSMs application.

The reconfigurable architectures in [102] is a directional construction in which no feedback loop exist. This construction rises naturally from the observation that many synthesis tools have problems with combinational loops when they are used to synthesize the programmable logic core along with the fixed part of the chip. The directional construction in [102] makes the signals flow between logic blocks flow from left to right only. To improve the hardware efficiency, a triangular construction is evaluated from the standard island-style architecture, also called rectangular construction. Results from [102] show that the triangular construction, called gradual architecture as well, performed better than the island-style architecture by 15% to 20%.

The success of the idea of directional construction helps guide the work in [103] in which a directional architecture is developed using PLAs as logic elements instead of 3-LUTs which is adopted as logic elements in [102]. The exploration of PLA-size discussed in [103] shows that PLAs with 9-input, 3-output and 9 or 18 product-term achieves 35% area improvements and 72% speed improvements over their LUT-based architectures from [102]. The experimental data also shows that a triangular gradual architecture performs better than a rectangular one for PLAs.

The routing architectures in both [102] and [103] are the novel one compared with traditional construction, in which all the logic elements are arranged in several levels. Take the routing architectures in [102] as an example, the outputs of one level PTBs can drive the inputs in all subsequent levels and final output ports. The great amount of multiplexer is employed in the routing network. In a 5-inputs application as shown in the architecture, there are eight 8:1 multiplexers in the second interconnection and four 12:1 multiplexers in third interconnection. The area of 8:1 multiplexer is 191.07 μm^2 based on the UMC 0.18 μm

CMOS technology library whereas it is only 16.262 μm^2 for a AND or OR gate targeted at the same library. In our opinion, lots of redundancies exist in the architecture.

V.5. Reconfigurable FSM Architectures

V.5.1 Reconfigurable FSM Architecture Overview

The proposed architecture is a hierarchical system which can be decomposed into different functional modules in 3 different levels. They are the sequential section and the logic section in the top level, sequential block and logic block in the medium level and PTBs in the base level. Each functional module is made up of sub-modules in following level**.** The capacity of architecture is the trade-off between application requirements and cost.

FSMs are usually used to describe the behaviour of digital circuits that transform sequences over one input into sequences over another (output) alphabet. So our proposed reconfigurable FSM is divided into two functional sections: logic and sequential, as shown in Figure V-12**.** The sequential block (Sequential B.) and logic block (Logic B.) are the functional units in their respective sections. These blocks are made up of PTBs.

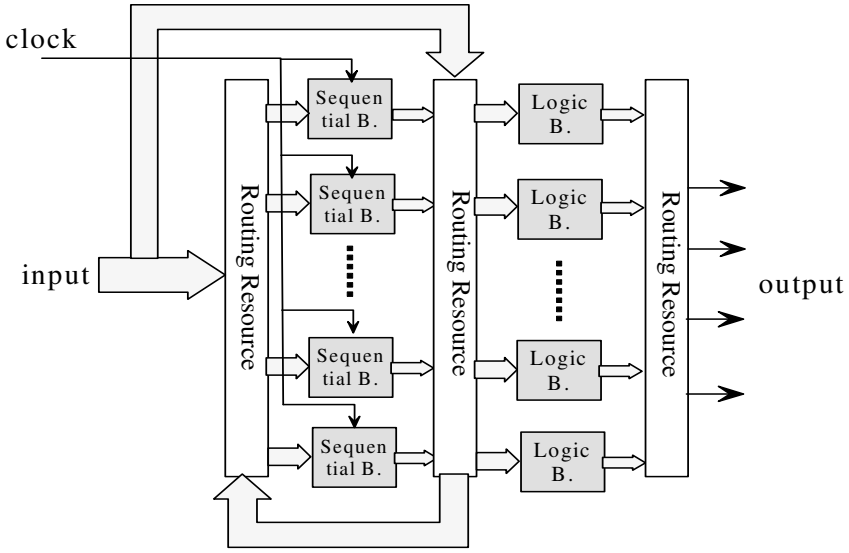

Figure V-12 : The architecture of a reconfigurable FSM

V.5.2 Functional Sections in the Architecture

The function of sequential section is to implement the transition between current state and next state. Outputs of this section respond to the current state of the FSM. The current state associated with the external input signals is used to compute the next states through certain Boolean equations. The changes of internal states from current state to next state will take place only if input clock is changed.

The function of the logic section is to implement any Boolean equation which can express the relationship among inputs, current state, next state and outputs. Outputs of this section are only dependent on the inputs and Boolean equation and do not change with time. The combination of logic block through interconnect switch matrix can realize any Boolean function.

The input signals which are up to 8 can reach both sections through the routing resources. The outputs of the sequential blocks can be routed to logic blocks and also fed back to other sequential blocks. There is no limit on the number of logic blocks which only depends on the number of FSM outputs. An architecture with 8 sequential blocks can implement a generic FSM with a maximum of 256 (2^8) states. The capability of the architecture can be extended by increasing the number of sequential and logic blocks with the extra cost in power, area and delay.

The current design size was derived from a benchmark set (LGSynth93 [104]) which is adopted by many researchers and will be discussed later. This benchmark set consists of 53 FSM test cases with the largest one containing 218 states. The current parameters of architecture make the architecture meet most of the applications' requirements with high efficiency in area, power and speed.

For a large FSM, decomposition is employed to overcome the limit on the size. The decomposition of FSM has been proved power saving [95-97]. In theory, any large FSM can be decomposed into a desired set of smaller sub-FSMs. Therefore, FSM of any size can be implemented with our architecture with a maximum number of 256 states after decomposition.

V.5.3 Architecture of Logic Block and Sequential Block

The logic block which consists of PTBs is a functional module in the logic section. The function of the logic block is to realize some combinational Boolean functions. A PTB acts as the basic computing unit to realize some basic Boolean functions. The architecture of logic block is shown in Figure V-13.

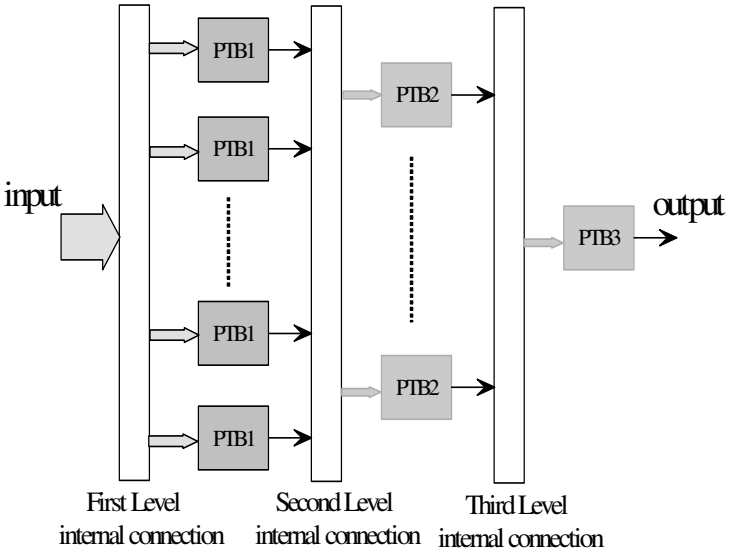

Figure V-13 : The architecture of a logic block

The PTBs are placed in several levels where a triangular architecture is adopted in which the number of PTBs in the second level is half of the first level and the number of PTBs in the third level is one fourth of its previous level. The outputs of the PTBs in each level can only reach the inputs of the PTBs in the next level.

Like the logic block, the sequential block is a functional module in the sequential section. Each sequential block is responsible for the changes in one state bit only. A sequential block can implement the combination of a Boolean function and transition between two different single bits which is synchronous with the input clock. A logic module followed by a D Flip-Flop is employed in the sequential section to realize the transition from current state to next state.

V.5.4 Construction of PTB

Product-term based style is adopted as a way to construct basic logic module in the architecture. A PTB unit consists of two sub-modules: the AND sub-module and the OR sub-

module. The AND sub-module can generate a product term. The output of AND sub-module is used for the OR sub-module. The OR sub-module is used to sum all the results from AND sub-modules and finally create the desired Boolean function result. It is easy to implement the AND sub-module and OR sub-module with the corresponding reconfigure bits and then construct different sizes of PTBs.

FSMs are usually too complex to easily map to a reasonable number of very fine-grained logic blocks [14]. In addition, they are too dependent upon single bit values to be efficiently implemented in a very coarse-grained architecture. Based on the required level of flexibility for the FSM, usually a mixture of fine and coarse grained architecture is used, except for some domain-specific functions. The architecture is made up of PTBs of different sizes which are assigned in their corresponding levels.

V.6. Low Power Implementation

Low power consumption is another key advantage of our architecture along with its reconfigurability. An unbalanced unsymmetrical architecture is adopted to reduce the area which contributes the most to the power saving. Based on the analysis of FPGA architectures, new strategies are drawn for interconnection network arrangement, construction of computing unit and the way of mapping PTB. More details are given in the following sub-sections.

V.6.1 A typical Interconnection Network

The most popular routing network is the balanced symmetrical construction. The functional modules are equally placed in relatively smooth sea of routing resources which **is** made up of switch boxes and connection blocks. This architecture is widely adopted in the commercial FPGA devices as shown in Figure V-14

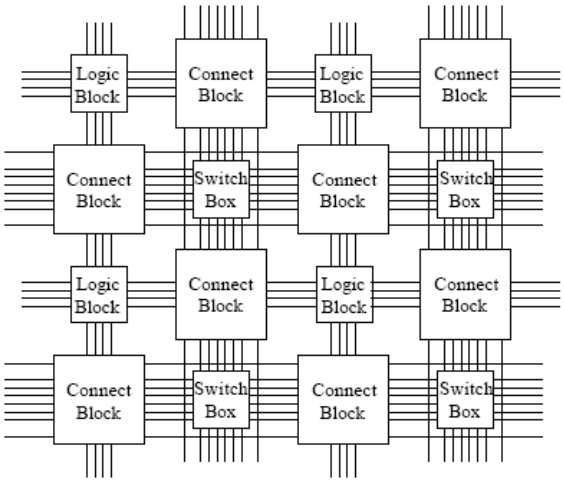

Figure V-14: A typical FPGA interconnection network

The overall reconfigurable architecture consists of functional modules (Logic Block) and routing elements (switch boxes and connection blocks). By employing programmable multiplexers, the connection blocks select the desired signal wire linked to the routing tracks. A connection block can attach the signals to the logic block and the switch box nearby. The connection in the switch boxes makes the input signal either pass through the switch box on its track or change its routing direction. Finally the signal reaches its destination logic block [14].

Obviously, the reconfigurability of FPGA devices is a result of the powerful interconnection network between the rows and columns of logic blocks. But there is also an area penalty to be paid for this flexibility. An example is given using 4-bit fully directional switch box in which all the ports in each side are bi-directional and the signal can enter the box through discretional port and reach any port for output. The area of such a switch box is 1433.025 μm^2 based on the UMC 0.18 μm CMOS standard cell library whereas it is only 12.197 μm^2 for an NAND gate targeted at the same library. This shows that a 4-bit fully directional switch box is equivalent to 117 NAND gates, which is enough to implement a small size FSM.

V.6.2 Interconnection Network

An efficient construction of the interconnection network is the key for solving the power and area problems. In some architectures, like FPGAs, where the logic blocks are embedded in

the routing network, interconnection network is made up of connection boxes and switch boxes; the area allocated to routing resources is over 80% of the whole system [105-107]. Therefore a high utilization ratio of interconnects will successfully reduce the area of the whole architecture.

The fine-grained blocks are useful for bit-level manipulations, while the coarse-grained blocks are well optimized for standard data path applications. Some architectures employ different sizes or types of blocks within a single reconfigurable array in order to efficiently support different types of computation. For example, memory is frequently embedded within the reconfigurable hardware to provide temporary data storage. Our design is targeted at general purpose applications, so only 1-bit interconnect in the architecture is considered. Mixed interconnects can be adopted if a special application is implemented.

The first reason why the architecture is called unbalanced is that the interconnects (switch box and fixed line) are arranged in an asymmetric manner.

Based on the area analysis of the routing resource area, our unbalanced architecture uses certain amount of fixed connection to replace the flexible switch boxes in order to reduce the area. There are three internal connection levels, as shown in Figure V-13. In the first level, multiplexer based switch boxes are used to feed the inputs of the logic computing modules. In this level, the internal connects keep their flexibility to make sure that input signals can reach their destination. With the guarantee of the required functionalities being implemented, the redundant connection paths are reduced to save area. In the second and third interconnection levels, most of the interconnections are made up of fixed connections which can greatly reduce the area compared with using switch boxes. However, a few multiplexers are kept within these two levels in order to keep the block flexible enough. Because the routing network area is reduced at the cost of flexibility, some redundant computing units are employed to compensate for the inconvenience of internal data exchange. But the area of the whole system is always kept at low level when compared with FPGAs.

V.6.3 Function of PTB

In our opinion, the Look-Up Tables (LUTs) used in FPGA architectures are redundant for FSM. A 4-input LUT is equally embedded in the sea of routing resource and is able to implement all possible 4-input logic functions with 16-bit configure bits. To implement a generic FSM, the sum of product terms is adopted as the basic way. Obviously, it is inefficient to implement a relatively fixed algorithm with the fully flexible components such as LUTs. So the keystone of our design is not only how to build an efficient and low area interconnects network but also to reduce the redundancy of logic computing modules compared with the ones in FPGA devices.

A two-input LUT, which can realize basic Boolean equations, is adopted as the basic unit to implement an AND sub-module or an OR sub-module. Compared with the computing unit in [108] whose area is 166.68 μm^2, the area of a two-input LUT is 73.17 μm^2 based on the same technology.

For large-sized PTBs, LUT is no longer a suitable way for implementation. As the number of inputs increases, the area of LUT will increase rapidly, leading to an exponentially increase in area usage. Take a four-input LUT as an example, the area of it is 3801 μm^2 based on the UMC 0.18 μm technology library, more than fifty times larger than a two-input LUT. Actually, it is unnecessary to implement PTBs with 4 or more inputs on fully flexible LUT. Since PTBs have limited functionality, the computing unit will not take full advantage of the flexibility of LUT, instead it will fully pay the cost of area and power consumption.

The combination of such computing unit, two-input LUT, in cascade and parallel mixed mode can construct PTB of any size.

V.6.4 Mapping of PTB

The number of PTBs in a logic block determines the efficiency of the block. A single PTB can be fully used. But the mass of data exchange between the logic blocks will increase the size of interconnects network and the effort for mapping and routing. So in this case, the whole system will not reach the best utilization efficiency. A large number of PTBs will

reduce the burden of interconnect switch matrix, but it will also increase the size of the total area. A balance is needed to resolve this problem.

The reason why the architecture is called unbalanced not only lies in arranging the interconnects (switch box and fixed line) in an asymmetric manner but also in selecting different type of PTBs for different levels.

We refer to the size of a PTB using the tuple (i, p, o) where i is the number of inputs, o is the number of outputs, and p is the number of product terms. To increase the capability of the architecture, (16,4,1) is selected for PTB1, (2,1,1) for PTB2 and (4,1,1) for PTB3. The adoption of PTB of different parameters is one of the reasons for adopting unsymmetrical interconnection mapping described in sub-section V.6.2. If PTB1 is selected for all levels, less efficiency in the second and third levels will increase the area of the whole system. For the same reason, PTB2 will result in less switch box efficiency in the first level and lead to the same outcome as in the previous case.

V.7. Experimental Results and Evaluation

Because similar domain-specific architecture for FSMs cannot be found in the literature, CPLD and FPGA devices are used as a reference to compare performance in area, power and delay with our architecture. Both devices are widely adopted in many designs and CPLD devices are especially suitable for implementing product-term applications.

The fair comparisons have been made, but some factors affect the comparison accuracy as described below:

1. FPGAs are well optimized for layout in physical level. Our architecture is synthesized and mapped to a standard cell technology library using standard ASIC automation tools. Therefore, the layout is not as optimized as in FPGAs.
2. In the process of mapping, ours is mapped manually which makes the best use of the special interconnection network and computing units. The mapping of FPGA and CPLD is performed with tools provided by Xilinx, Inc. Compared with manual mapping, the tools cannot achieve the best utilization of dedicated connection in FPGA which will reduce the delay time and power consumption.

V.7.1 Experimental Platform

The synthesis tool for our reconfigurable architecture is Ambit BuildGates from Cadence Design Systems, Inc. The architecture is targeted at the UMC 0.18 μm three-metal CMOS technology library. The area and delay time is obtained from the synthesis. The power consumption values for our reconfigurable architecture are obtained after post-layout simulation by Synopsys PrimePower. The synthesis and mapping tools for FPGA and CPLD devices are ISE V6.2i of Xilinx, Inc. Their power consumption data is obtained with XPower.

According to the library and voltage in the synthesis platform used for our architecture, one typical device is adopted in FPGA and CPLD category respectively: xcv50e-6cs144 [10], the smallest device in the FPGA Virtex-E family; xc2c128-4vq100 [109], the device has the capability of 448 product terms in the CPLD CoolRunner-II family, whose scale is the most close to our architecture. These two devices are used in comparisons in area, power consumption and delay with our architecture.

Table V-2: Test cases and their characterizations

Name	I	O	P
lion	2	1	11
dk27	1	2	14
dk512	1	3	30
s27	4	1	34
tav	4	4	49
bbara	4	2	60
dk16	2	3	108
planet	7	19	115
S1488	8	19	251
tbk	6	3	1569

Several commonly used test cases from LGSynth93 [104] have been implemented on CPLD, FPGA and our reconfigurable array. The benchmark set is adopted by many researchers [103, 110, 111] who work on implementing FSMs. The FSMs in the benchmark are all Mealy machines except for Cnt32 and Cnt64 which are Moore type. The test cases in the benchmark cover the most typical applications. From this benchmark set, ten test cases including the biggest one (tbk) were selected, all with different degrees of complexity and a

number of inputs and outputs, as shown in Table 1. We describe the characteristic of FSM with the tuple (I, O, P) where I is the number of inputs, O is the number of outputs, and P is the number of product. The maximum input, output and state number among the test cases is 8, 19 and 48 respectively. The selected cases will be implemented on FPGA, CPLD and our architecture for the comparisons.

V.7.2 Experimental Data Pre-process

For different FPGA families, various hardware resources provide improved performance in delay and power consumption. In VirtexE family, the architecture has dedicated connections between adjacent LUTs allowing them to be connected without using the switchboxes in the general routing matrix [10]**.** Virtex-II and later families also have dedicated connections and OR gates for implementing large sum-of-products expressions [112]**.**

Although the smallest device is chosen in the comparison, the utilization rate of the FPGA device is very low. In order to study how the size of devices affects the area and power consumption, different chips are selected to perform area and power comparisons when the same testcase is implemented on them.

It is well known that different CMOS technologies will lead to different results in area, power and delay. It is very difficult to compare the power consumption and delay time between distinct family devices. Therefore, the delay time and power consumption comparisons are made between the devices using the same family with the same technology to avoid the effects of different technology. The number of occupied Slices is adopted for area comparison, which indicates the hardware utilization but does not give information about the CMOS technology used.

Twenty eight FPGA devices from seven families are selected to study how the area occupation is affected by the utilization rate and the different dedicated hardware resources, as shown in Table V-3. The smallest and largest devices are selected from each family with the maximal area ratio among them being 200 (xc2v40 vs. xc2v8000). The same benchmark is implemented on all devices to obtain the hardware utilization rate which is derived from the number of LUTs and the number of occupied Slices in the implementation summary.

Table V-3: Selected FPGA devices

Spartan-II	Spartan-IIE	Spartan-3/3L	Virtex
xc2s15	xc2s50e	xc3s50	xcv50
xc2s100	xc2s200e	xc3s1000	xcv400
xc2s200	xc2s600e	xc3s5000	xcv1000
Virtex-E		**Virtex-II**	**Virtex-4**
xcv50e xcv100e		xc2v40	xc4vfx12
xcv200e xcv400e		xc2v500	xc4vfx40
xcv600e xcv1600e		xc2v2000	xc4vfx80
xcv2600e xcv3200e		xc2v8000	xc4vfx200

Two test cases are implemented on FPGA devices. One small testcase is chosen in order to give a low utilization rate on the smallest device (Virtex-E xcv50e), and another large testcase is chosen, so that it gives high (97%) utilization rate on the same device.

For the small testcase, the number of used LUTs for all devices in one family is the same. Also it is the same for all families except for Virtex-4 family. For the large testcase, the same situation can be observed when the utilization rate is lower than 90% and the difference is less than 1% in the case of over 90% utilization rate.

Clearly, the area occupation is slightly affected by the chip utilization rate, regardless of the size of the FPGA device, when it is scaled by occupied number of LUTs in FPGA devices.

To quantify the degree that different FPGA devices affect the delay time, the large testcase is implemented on 8 devices in Virtex-E family. The delay time is the same when the same testcase is implemented on different devices. Clearly, device size has no impact on the delay time.

Because the utilization rate of the selected FPGA device (Virtex-E xcv50e) and our architecture are 12% and 60% respectively for case tbk, it's estimated that FPGA device is 5 times larger than our architecture. Therefore, the FPGA device pair with the same proportion in size is taken as the reference to reveal how the size affects power consumption.

To find out the appropriate FPGA device pairs, all devices in VirtexE family are listed in Figure V-15. The size of smallest device, xcv50e, is scaled as the basic unit. All the device sizes are scaled as the multiples of the unit. The column of 5X times size of each device follows as the reference frame as shown in Figure V-15 until the 5X times size of the device xcv600e is larger than the biggest one in the family.

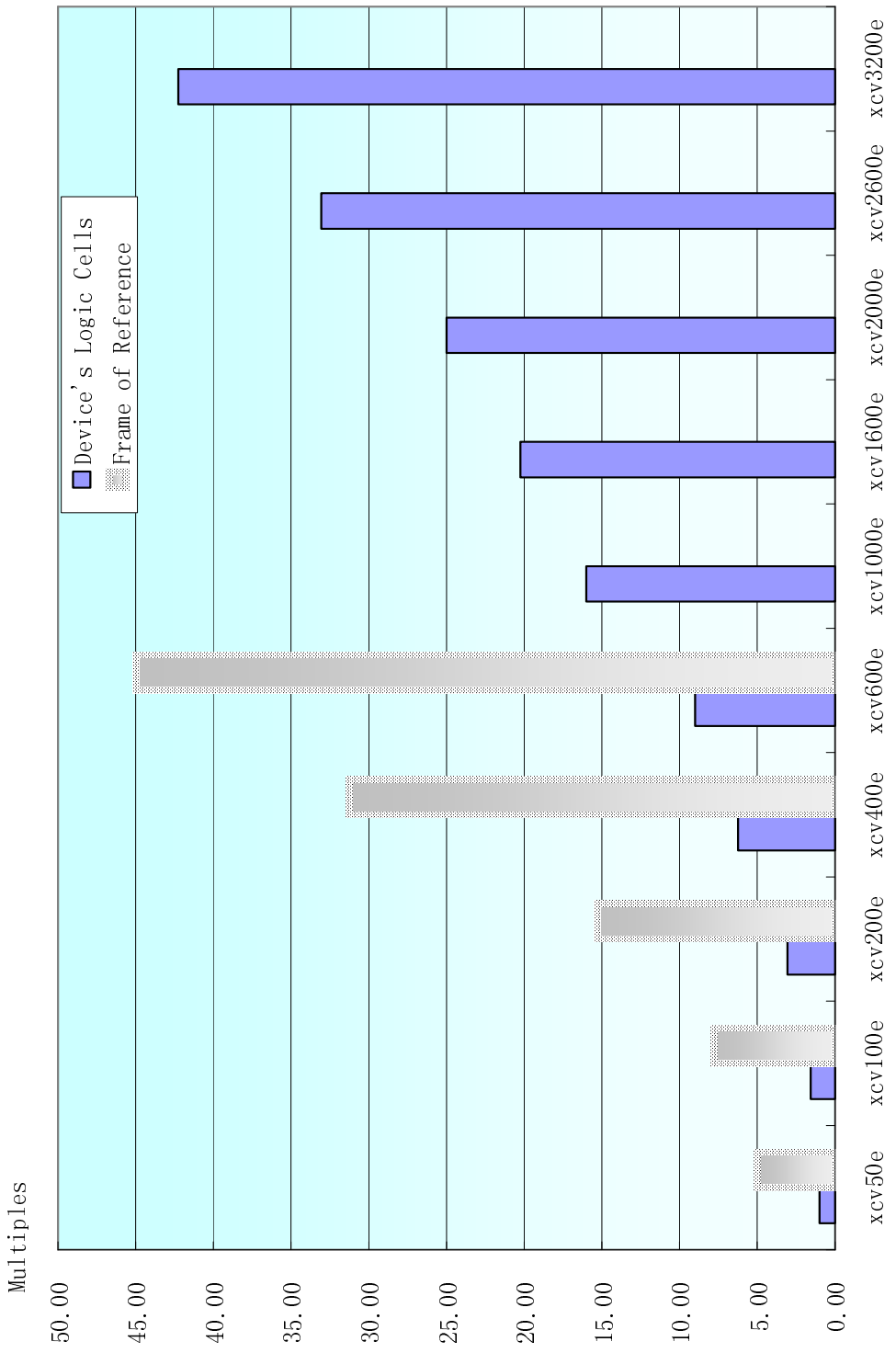

Figure V-15: The illustration of comparison in size between FPGA devices

From Figure V-15**,** it is easy to find out 5 device pairs whose power consumptions are shown in Figure V-16.

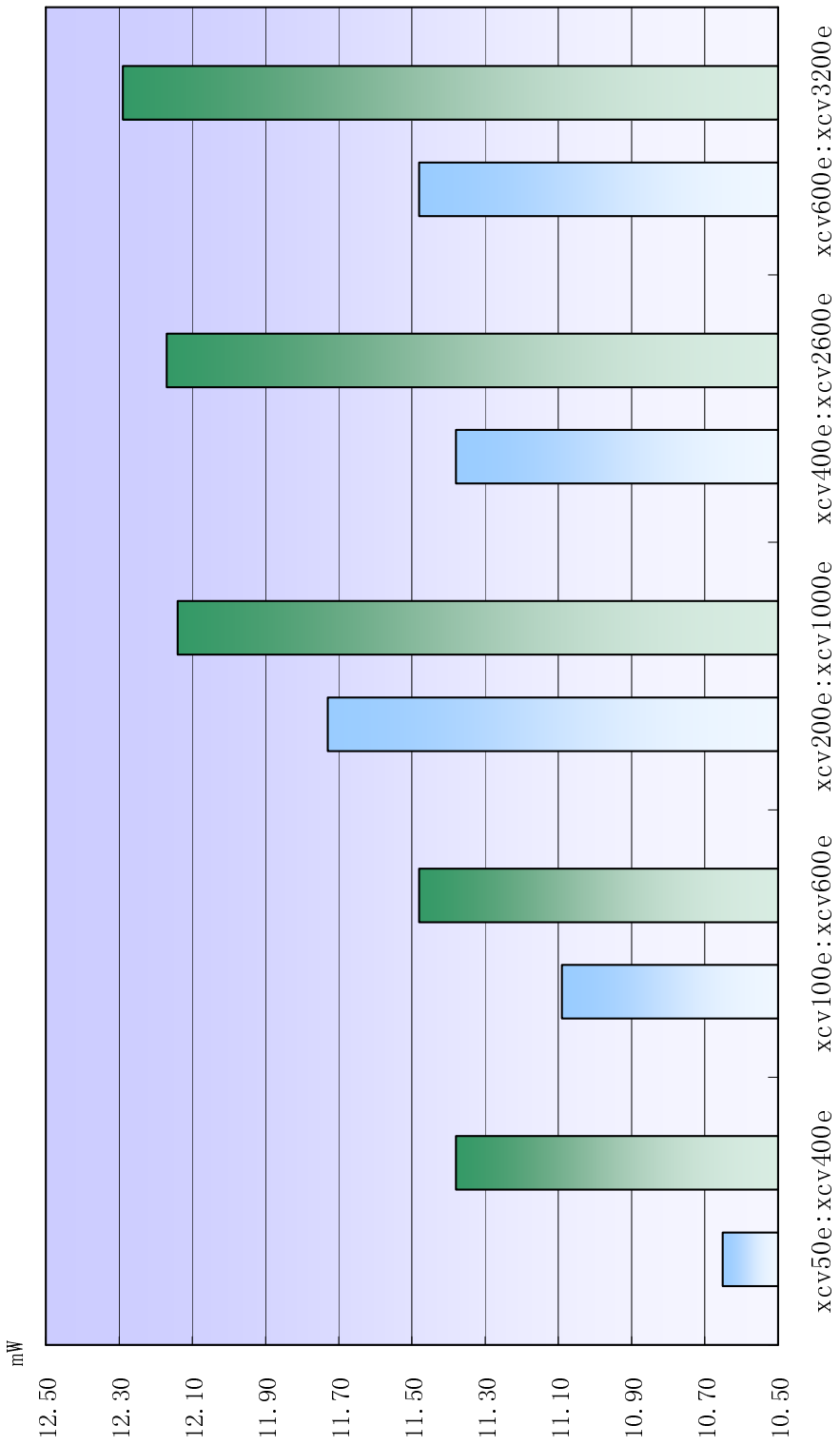

Figure V-16: The power consumption comparison of FPGA device pairs

Generally, the power consumption increases when the device size increases. From Figure V-16, the difference in scale will lead to about 10% increase in power consumption. In the following sub-section, this rate is used to process the raw experimental data.

V.7.3 Power consumption Comparison

Power consumption of an FPGA device falls into two parts: Vccint is consumed by the core inside the device, and consists of clock power consumption, inputs power consumption, logic power consumption and signals power consumption; while Vcco is consumed by the I/Os. Both parts can also be classified as quiescent and dynamic. Quiescent power is the power consumed with no switching. Charging and discharging of parasitic capacitances cause dynamic part.

Because the target FPGA device is larger than our architecture, quiescent power consumed by unused computing units, RAM, clock manager and controller in FPGA device are very high. To remove all these impact factors, only the Vccint dynamic power consumption is taken as the reference, as shown in Table V-4**,** column four.

Table V-4: Experimental results for power consumption

Name	ASIC (10^{-2}mW)	Our Archi. (mW)	FPGA Vccint (mW)	FPGA (Scaled) (mW)	CPLD (mW)
lion	0.326	0.181	0.87	0.78	0.6
tav	0. 729	0.253	1.17	1.05	1.41
s27	0. 494	0.239	1.23	1.11	0.94
dk27	2.547	0.38	1.94	1.75	2.16
dk512	2.352	0.242	0.95	0.86	1.13
bbara	1.076	0.305	2.25	2.03	1.32
tbk	2.555	0.434	2.01	1.81	1.49
dk16	7.085	0.256	1.39	1.25	1.57
planet	7.071	0.517	3.9	3.51	5.26
S1488	14.521	0.542	5.78	5.2	5.68

The scaled FPGA power consumption, listed in the fifth column in Table V-4, is obtained by scaling down the power in the second column by 10% in order to remove the impact of device size. Our architecture's power consumption is listed in the third column.

The target CPLD device has a similar capability of mapping product terms to our architecture. The dynamic power consumption of the CPLD is listed in the sixth column in Table V-4.

In order to simplify Table V-4 and make their comparative relations clearer, Table V-5 is listed below. In this table, the amount of FPGA scaled power consumption and CPLD power consumption are represented as "times" relative to the power consumption amount of our architecture.

Table V-5: Normalized power consumption of FPGA and CPLD

Name	FPGA (Scaled) (times)	CPLD (times)
lion	4.3	3.3
tav	4.2	5.6
s27	4.6	3.9
dk27	4.6	5.7
dk512	3.6	4.7
bbara	6.7	4.3
tbk	4.2	3.4
dk16	4.9	6.1
planet	6.8	10.2
S1488	9.6	10.5

The comparisons made with FPGA scaled power consumption and CPLD power consumption are listed in Figure V-17 and Figure V-18 respectively.

Clearly, our architecture achieves from 71.9% to 89.6% power savings compared with the FPGA Vccint dynamic scaled power consumption. Compared with CPLD dynamic power consumption, our architecture saves power consumption by 70.9% to 90.5%. Obviously, our architecture achieves a good performance in power consumption.

It is noticed that FPGA device consumes more power in some cases, while CPLD device consumes more in other cases. It is difficult to say that one consumes more power than the other between FPGA and CPLD devices. Basically, a CPLD device consumes more power than a FPGA device when the number of outputs is higher than the number of inputs.

V.7.4 Area & Delay Comparison

All test cases were implemented with our reconfigurable architecture, CPLD and FPGA device for comparison. CPLD is course-grained device based on product-term/macrocell technologies with lower density. For the same capability, the area of CPLD device is larger than FPGA device. FPGAs are usually having more gates in a given area and cost less than their CPLD cousins. Therefore, area comparison is made only between FPGA device and our architecture.

The area results are listed in Table V-6**.** The area estimation of Virtex-E is based on two LUTs per slice where an estimated area of 3303 μm^2 is used per slice and its surrounding routing [73]. This value excludes the area of memory which is embedded in FPGA device.

Table V-6: Experimental results for area and delay

Name	Area (μm^2)			Delay (ns)		
	ASIC	FPGA	Our Re. Archi.	Our Re. Archi.	FPGA	CPLD
lion	317	13212	7185	4.87	9.27	5.70
tav	358	16515	9649	4.87	13.02	5.70
s27	407	33030	15005	4.87	11.07	5.70
dk27	610	16515	10297	4.87	8.01	3.40
dk512	1167	42939	21841	4.87	12.52	5.70
bbara	1187	46242	28052	4.87	10.3	5.70
tbk	4415	310482	180325	4.87	20.52	8.00
dk16	3638	102393	52166	4.87	15.29	5.70
planet	7891	330300	153214	4.87	22.62	11.00
S1488	9526	640782	284704	4.87	25.08	10.00

The delay time listed in Table V-6 is the period from the rising edge of the clock to the moment when signal reaches the output port, namely, clock path plus data path. The CPLD delay time is obtained through a timing simulation after the HDL design has been synthesized, placed and routed.

In order to make the relations of our architecture with FPGA and CPLD more recognizable, Table V-7, the simplified form of Table V-6, is listed below in which the area and delay of FPGA and CPLD are normalized and represented as “times” based on our architecture.

Table V-7: Normalized area and delay of FPGA and CPLD

Name	Area (times)	Delay (times)	
	FPGA	FPGA	CPLD
lion	1.8	1.9	1.2
tav	1.7	2.7	1.2
s27	2.2	2.3	1.2
dk27	1.6	1.6	0.7
dk512	2.0	2.6	1.2
bbara	1.6	2.1	1.2
tbk	1.7	4.2	1.6
dk16	2.0	3.1	1.2
planet	2.2	4.6	2.3

It can be seen from Figure V-17 that our reconfigurable architecture achieves from 37.7% to 55.6% improvements in area compared to Virtex-E. This result proves our strategy for reducing routing network which leads to significant saving in area, that is, a smaller area contributes to significant power saving.

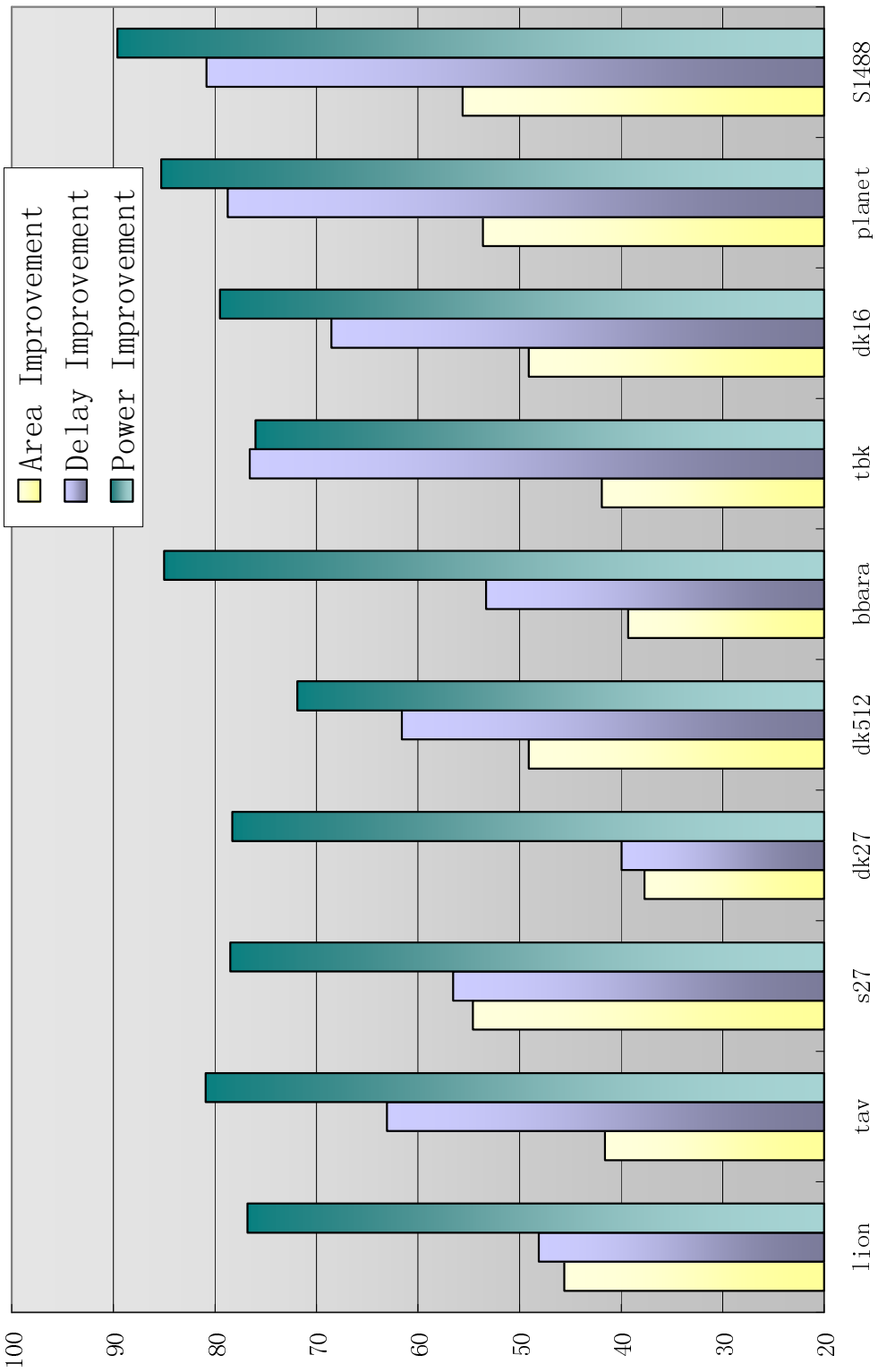

Figure V-17 : The improvements compared with the FPGA device

Because some fixed lines are adopted to save area, the flexibility of the architecture is reduced. This makes the critical path from inputs to outputs almost the same for different cases. The same critical path results in a constant delay time for all the test cases as shown in Table V-6. It can be seen that the delay time of our architecture is smaller than the FPGA

and CPLD devices except in the case of dk27. For all the test cases, the delay time of the CPLD device is about 50% smaller than the FPGA device.

The comparisons of delay made with FPGA and CPLD devices are listed in Figure V-17 and Figure V-18 respectively. Compared with FPGA device, our architecture achieves from 39.95% to 80.82% reduction. The average delay time of our architecture for all test cases is 20% shorter than CPLD device. It needs to be emphasized that this improvement is obtained together with significant reduction in area occupation and power consumption.

From the comparison in area, power and delay between our architecture and PLD (FPGA and CPLD) devices, it is clear that we use a compact architecture to implement FSM with less occupied area, less power consumption and shorter delay time. The small hardware and short critical path directly leads to saving on power.

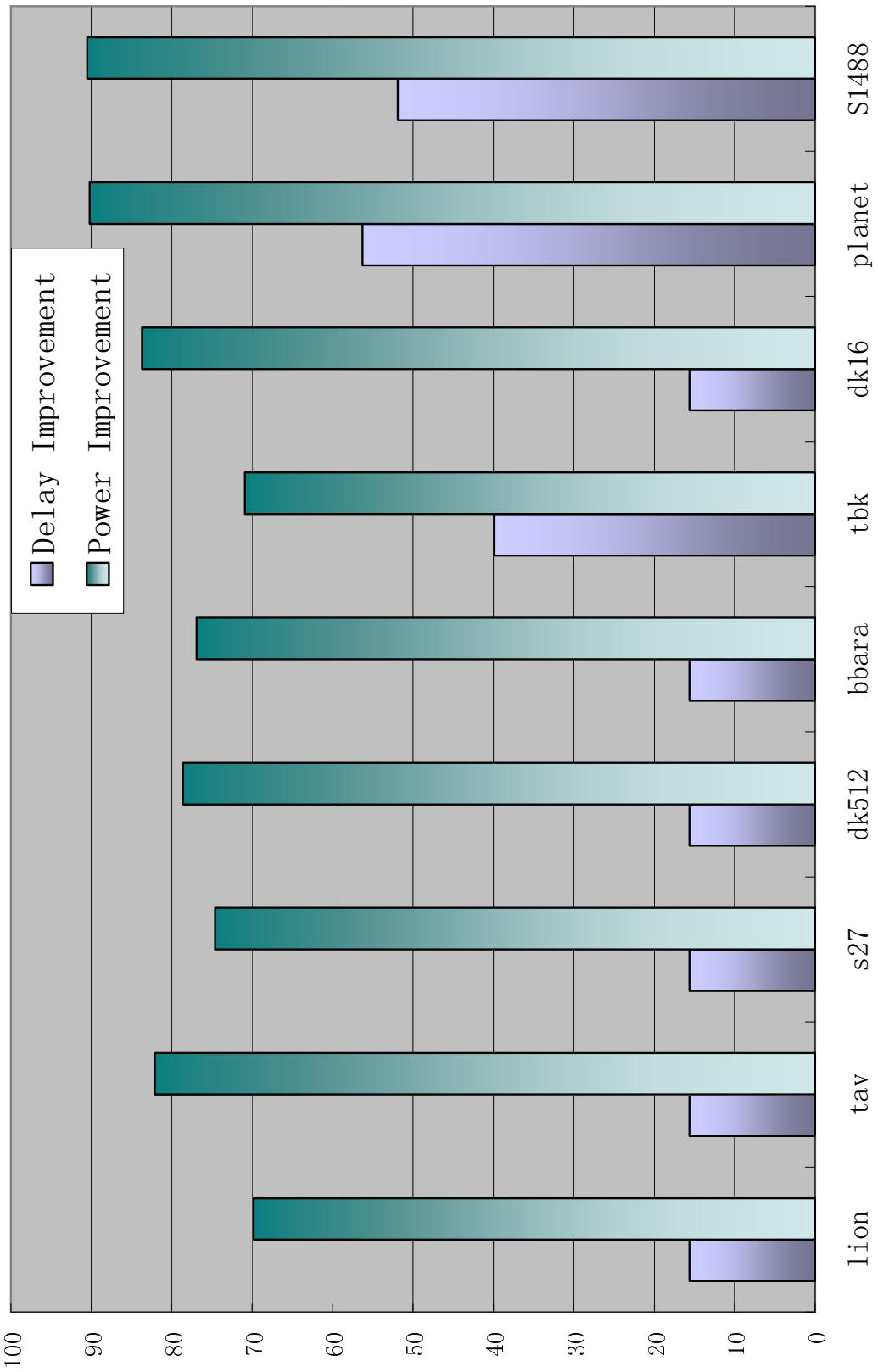

Figure V-18 : The improvements compared with the CPLD device

V.7.5 Power Consumption, Area and Delay after Decomposition

Because our architecture has limit on capability, FSMs decomposition is adopted frequently for large cases and low power implementations. A test is performed in order to obtain the performance in power, area and delay. Even the biggest testcase in the benchmark set can be implemented with our architecture; therefore, test cases planet and S1488 are used as the two sub-modules to present a case when a large FSM is decomposed. These two sub-modules can also be decomposed into smaller modules as desired.

In the power consumption test, the rate of power-saving depends on the number of sub-modules employed, the size of each sub-module and the working time of sub-modules. The experimental data clearly shows that decomposition reduces power consumption at the cost of more area occupation. Because of the special architecture, the delay time is fixed even when more sub-modules are decomposed.

Obviously, decomposition will not increase power consumption and delay time for the cases of large FSMs. The cost for large FSMs is the large occupied area which depends on FSM size. In other words, the larger the FSM, the more the area. The results show that the area of our architecture will be significantly smaller than an FPGA device for the case of large FSMs.

V.7.6 Relationship between Power, Area and Delay

All the comparison of improvements on area, delay time and power are list in Figure V-17 and Figure V-18 respectively. Basically, our architecture achieves good performance in power, area and delay in the case of large ones. For our architecture, each case is fully optimized in mapping and routing and achieves the best utilization rate. However, the FPGA mapping and routing performed by the tools are not the most optimized ones and can be improved in the case of large benchmarks. It is an important reason for better achievements of our architecture for large cases.

In the case of s27, a small size benchmark, the area improvement is a bit lower than the best case whereas its power consumption just reaches the average level. This is because the case just fits our architecture well. Because of the amount replacement of fixed lines in the

interconnection network, some PTBs in three levels are cascaded to form a fixed path. In most cases, all PTBs in one fixed path can be fully utilized. But there is an exception for case s27, in which the hardware resource reaches the top utilization rate. For the power consumption experiments, the major cases consumed more power when they are implemented in CPLD than FPGA devices. The exceptional ones are the small cases in which the number of outputs is smaller than the number of inputs.

V.8. Conclusion

A novel reconfigurable low-power domain-specific FSM architecture for control purpose has been introduced in this chapter. Based on the analysis of traditional interconnection networks, a reduced one is adopted at the cost of less flexibility in order to improve area efficiency. The new product-term based computing units are employed to implement the basic Boolean function. The unsymmetrical design style is used in arranging interconnection, the selection of PTBs and the way of mapping basic computing units.

The reduced flexibility of interconnection and basic computing units in the new architecture achieves a significant reduction in area which directly leads to lower power consumption. For this reason, unlike commercial FPGA devices, the proposed architecture targeted at generic FSMs is not flexible enough to be used in any application. Obviously, the architecture suitable for any application will be a large construction with more redundant parts which will lead to more area and power consumption. One of the purposes of our architecture is to find a fine balance between size and power consumption.

Ten test cases from the widely adopted FSM benchmark set were implemented by using both our architecture and a FPGA device. It was demonstrated that our architecture could obtain an average reduction of 82% in power consumption, a decrease of 44% in area occupation and 20% reduction in delay when implementing the same circuit on a commercial FPGA device. These figures show that the proposed reconfigurable architecture for FSMs provides an efficient hardware platform for the implementation of generic FSMs in various power-sensitive designs. The flexibility of the architecture makes it convenient not only for the

proposed processor in this dissertation but also for embedding them in any reconfigurable SoC in various applications such as mobile devices, portable players, etc.

Chapter VI

Implementation of DA Application

VI.1. DCT Implementation

DCT is one of the most popular and effective compression coding scheme and can be found in almost all standardized video coding algorithms such as ITU H.261, H.263 and H.264 for video conferencing standard, and ISO MPEG (including MPEG-1, MPEG-2, and MPEG-4) for visual communication and multimedia applications. The broader use of DCT in communication and multimedia areas underlines the requirement for a more efficient and flexible system.

MPEG4 is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group), and became an International Standard in 1999. Compared to its predecessor MPEG2, MPEG4 greatly improves the perceptual video quality by introducing some new tools in the encoding and decoding process. A wide range of applications are supported, from 5-64k bits/s for mobile video to 2M bits/s for TV/film applications. Nine profiles are defined in MPEG4, of which Simple Profile (SP) is suitable for mobile video applications. The prediction errors are DCT transformed and quantized. The control data, quantized prediction errors, and motion data are encoded in the Entropy Coding module, which is then packed into video streams for transmission or storage.[113]

H.264 is a joint effort between ITU-T Video Coding Experts Group (VCEG) and ISO/IEC Motion Picture Experts Group (MPEG), and became an International Standard in 2003.

The new key features of H.264 include:

- Enhanced Motion Estimation
- Small blocks for transform coding

- 4x4 Integer transform
- In-loop de-blocking filter
- Enhanced entropy coding CAVLC (Context Adaptive Variable Length Coding)

The H.264 Codec design substantially increases the complexity (memory & computation), requiring approximately 3x computation power for the decode and 4x for the encode compared to MPEG-2 Codec design. Three profiles are provided in H.264: I) Baseline profile; ii) Main profile; iii) Extended profile. Of the three, Baseline profile may be adopted for mobile application. [114]

VI.1.1. DCT Algorithm

For a given 2-D spatial input vector $\{X_{i,j};i,j=0, 1, \ldots, N-1\}$, the 2-D DCT output vector $\{ Y_{k,l};k,l=0, 1, \ldots, N-1\}$ is defined as follows:

$$Y_{k,l} = C_k C_l \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \cos\frac{\pi k(2i+1)}{2N} \cos\frac{\pi l(2j+1)}{2N} X_{i,j} \qquad \textbf{(VI-1)}$$

where

$$C_x = \begin{cases} 1/\sqrt{N} & x = 0 \\ 1/\sqrt{2/N} & 1 \le x \le N-1 \end{cases} \qquad \textbf{(VI-2)}$$

Employing row-column decomposition, 2-D DCT is separable and can be broken into two sequential 1-D DCT operations, one along the row vector and the other along the column vector of the preceding row vector results. Therefore, 1-D DCT implementation is targeted at the proposed architecture.

For an input vector $\{X_0, X_1 \ldots X_{N-1}\}$, the 1-D DCT output vector $\{Y_0, Y_1 \ldots Y_{N-1}\}$ is given as follows:

$$Y_k = C_k \sum_{i=0}^{N-1} \cos\frac{\pi k(2i+1)}{16} X_i \qquad \textbf{(VI-3)}$$

where

$$C_k = \begin{cases} 1/\sqrt{N} & k = 0 \\ 1/\sqrt{2/N} & 1 \le k \le N-1 \end{cases} \quad \textbf{(VI-4)}$$

For eight points 1-D DCT, we define coefficient matrix F_k as:

$$F_k = \left[F_k(i)\right] = C_k \cos\frac{\pi k(2i+1)}{16} \quad \textbf{(VI-5)}$$

where i = 0,1……7; The coefficient matrix includes the C_k and cosine factors. Then, Equation (VI-3) can be rewritten as:

$$Y_k = \sum_{i=0}^{7}\left[F_k(i)\right]X_i = \sum_{i=0}^{7} F_k X_i \quad \textbf{(VI-6)}$$

Now, the equation of DCT has been transformed into the format as shown in Equation (III-10). The next step is to find the common terms sharing of DCT to maximize the hardware efficiency.

VI.1.2. 2-D DCT and its Implementations

To verify the functionality of the reconfigurable architecture, an 8 bits 2-D DCT is implemented. In the realization of the DCT, finite accuracy is achieved due to fixed DA precision. Obviously, more accurate data can be obtained through increasing the precision of the coefficients and the width. This, however, results in larger area and higher power consumption, and adversely affects the computing speed in the adder array.

The requirements of DCT and Inverse Discrete Cosine Transform (IDCT) hardware implementations are imposed by various standards, such as ISO/IEC 14496-2:2004 and IEEE Std 1180–1990 [115]. A brief summary is given below:

- Image pixel representation: 8 bits for 8X8 DCT
- Input bits for the forward transform: 9 bits
- Coefficients representation: 12 bits
- 1-D DCT outputs: 14 bits

There are many papers discussing the fast implementation of the 1-D and 2-D DCT/IDCT. Based on the straightforward implementations in [116-118] which are computationally expensive with 4096 multiplications, the fast algorithms reduce the computational cost. Theses algorithms can be broken down into two broad categories: one is the so-called indirect method based on the row-column decomposition [119-122] and the other is direct, fast 2-D approaches [123-126]. The row-column approach results in simple and regular implementations, but it is less computationally efficient than direct, fast 2-D implementations.

The row-column algorithm is divided into three main stages. Stage one and stage three compute the row and column transforms, respectively. In stage one, the one–dimensional (1-D) DCT/IDCT of each row of input data is taken, and these intermediate values are transposed. Then, in the stage three, the 1-D DCT/IDCT of each row of the transposed values is fed to the 2-D DCT/IDCT in column. Both the row and column transforms are implemented using the same 1-D DCT module shown in Figure VI-1. The second stage performs the transposition using N^2 registers where N is 8 in our implementation.

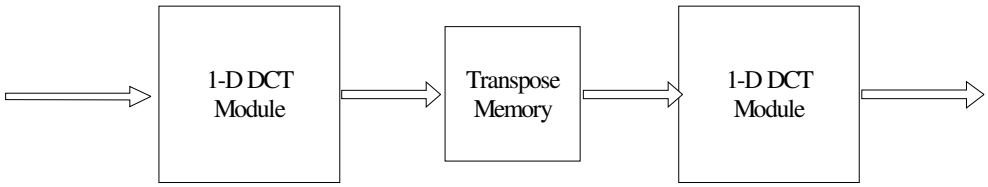

Figure VI-1 : A general row-column 2-D DCT implementation

VI.1.3. Control Path Implementation

The control unit in the proposed DA processor can be considered as a big multi-input-multi-output FSM. This FSM can be divided into some separated sub-module which can perform specific function independently.

A full row-column 2-D DCT requires two 1-D DCT modules along the data path. According to our architecture, only one algorithm logic unit is inside, and the whole 2-D DCT must be broken into two separated 1-D DCT stages. In these two stages, the algorithm logic unit runs twice and the registers matrix will be used for storing data temporarily. Registers matrix is

adopted not only because a whole 2-D DCT data path is broken into two separated parts. Based on the definition of 2-D DCT in Equation (VI-1), the 8 parallel data input to the second 1-D DCT module are not the ones obtained directly from the results of the first 1-D DCT module. These 8 data are vertical to the results from the first 1-D DCT module. Therefore, the inputs to the second 1-D DCT module will not be available until 8 results from 1-D DCT module are ready. If the results from 1-D DCT module are stored row-by-row, as shown in Figure VI-2, from Row_0 to Row_7, the data fed to the second 1-D DCT module will be column-by-column from Col_0 to Col_7 and vice versa.

Row_0	Reg0	Reg1	Reg2	Reg3	Reg4	Reg5	Reg6	Reg7
Row_1	Reg8	Reg9	Reg10	Reg11	Reg12	Reg13	Reg14	Reg15
Row_2	Reg16	Reg17	Reg18	Reg19	Reg20	Reg21	Reg22	Reg23
Row_3	Reg24	Reg25	Reg26	Reg27	Reg28	Reg29	Reg30	Reg31
Row_4	Reg32	Reg33	Reg34	Reg35	Reg36	Reg37	Reg38	Reg39
Row_5	Reg40	Reg41	Reg42	Reg43	Reg44	Reg45	Reg46	Reg47
Row_6	Reg48	Reg49	Reg50	Reg51	Reg52	Reg53	Reg54	Reg55
Row_7	Reg56	Reg57	Reg58	Reg59	Reg60	Reg61	Reg62	Reg63
	Col_0	Col_1	Col_2	Col_3	Col_4	Col_5	Col_6	Col_7

Figure VI-2 : Registers matrix

Therefore, input matrix will route 8 external signals and the 8 results from the first 1-D DCT module to algorithm logic unit in turn. The reconfigurable FSM controlling input matrix is configured as a 4-bit counter and the highest bit of the counter which outputs 0 and 1 alternatively is used to control the two-input multiplexer to switch the input port of algorithm logic unit between external signals and the results from the first 1-D DCT module.

Similar to input matrix, output matrix will export 8 valid results serially every 8 clock cycles and the reconfigurable FSM in output matrix is configured as a 4-bit counter as well. The highest bit of the counter is used to control output buffer refreshing or not: in case of value of 0, output buffer keeps its original value; in case of value of 1, output buffer is refreshed in

each clock cycle by data that arrive.

VI.1.4. Registers Matrix Implementation

The registers matrix in proposed processor has two working modes: one dimension 64 registers and 8X8 two-dimension array. In considering the 8 parallel outputs from algorithm logic unit, the registers matrix is configured as an 8X8 two-dimension array. It means that only two 3-bit reconfigurable FSMs in row and column control modules are used to address coding in implementation of 8X8 2-D DCT.

To make the FSM in row control module work properly, a reconfigurable FSM is set as a 4-bit counter whose initial value is set as binary '1000'. The highest bit of the counter is connected to the enable port of row control FSM. Along with the inverter which is available only in 8X8 two-dimension working mode and bridges the two enable ports of row and column control FSMs, the column address coding FSM will operate every 8 clock cycles when the address coding for rows is done. The two reconfigurable FSMs in row and column control unit are configured as a 3-bit counter, one of the simplest FSM of all implementations.

Under the control of enable port, 3-bit counter in row or column control unit works independently generating address from 0 to 7 in 8 clock cycles; the counter in row control alternates with the one in column control.

VI.1.5. Algorithm Logic Unit Implementation

A. Coefficient Matrix and Terms of DCT

The precision of DCT implementation lies in the coefficient representation when the input vector is given with the fixed precision. To fully support international standards such as ISO/IEC 14496-2:2004 and IEEE Std 1180–1990, the coefficient precision in our architecture is set as 12 bits. Following the steps of adder-based DA, Equation (VI-5) can be represented in 2's complement format as shown in Figure VI-3, where coefficient matrixes $F_0(i)$, $F_1(i)$,…, $F_7(i)$ are listed.

$$F_0(i) = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix} \quad F_1(i) = \begin{vmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{vmatrix}$$

$$F_2(i) = \begin{vmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{vmatrix} \quad F_3(i) = \begin{vmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{vmatrix}$$

$$F_4(i) = \begin{vmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{vmatrix} \quad F_5(i) = \begin{vmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{vmatrix}$$

$$F_6(i) = \begin{vmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{vmatrix} \quad F_7(i) = \begin{vmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{vmatrix}$$

Figure VI-3 : $F_k(i)$ in 2's complement format

The eight coefficient matrixes can be converted to the terms in the format of input vector $\{X_0, X_1 \ldots X_{N-1}\}$. In theory, 96 (=12X8) terms, which are the summation of eight inputs, are needed for 8-point 1D DCT with 12-bit coefficients based on the adder-based DA. This implies that 672 two-input adders are required when implemented directly. However, because of the periodic conjugate symmetry inherent in the DCT, the real implementation consumes just a small part of the theoretic hardware cost. In total, there are 96 terms of eight coefficient matrices. In deducting the zero and duplicate terms which need no further calculation, there are one term of 8 inputs and 22 terms of 4 inputs. The eight coefficient matrixes in format of input vectors and set R which contains all non-repetitive terms are shown in Table VI-1 and Table VI-2 respectively.

Table VI-1 : Eight coefficient matrixes in format of input vectors

$F_0(i)$
0
0
$X_0X_1X_2X_3X_4X_5X_6X_7$
0
$X_0X_1X_2X_3X_4X_5X_6X_7$
$X_0X_1X_2X_3X_4X_5X_6X_7$
0
$X_0X_1X_2X_3X_4X_5X_6X_7$
0
$X_0X_1X_2X_3X_4X_5X_6X_7$
0
0

$F_1(i)$
$X_1X_3X_5X_7$
$X_1X_3X_5X_7$
$X_0X_3X_5X_6$
$X_0X_2X_4X_6$
$X_1X_2X_4X_7$
$X_0X_2X_4X_6$
$X_0X_1X_3X_5$
$X_0X_3X_5X_6$
$X_0X_1X_4X_5$
$X_0X_1X_2X_4$
$X_4X_5X_6X_7$
$X_4X_5X_6X_7$

$F_2(i)$
$X_1X_3X_4X_6$
$X_0X_1X_6X_7$
$X_1X_3X_4X_6$
$X_2X_3X_4X_5$
$X_0X_2X_5X_7$
$X_0X_2X_5X_7$
$X_2X_3X_4X_5$
$X_0X_1X_6X_7$
$X_0X_1X_6X_7$
$X_0X_2X_5X_7$
$X_2X_3X_4X_5$
$X_2X_3X_4X_5$

$F_3(i)$
$X_0X_2X_3X_6$
$X_0X_2X_3X_6$
$X_3X_5X_6X_7$
$X_1X_4X_6X_7$
$X_0X_1X_2X_4$
$X_1X_4X_6X_7$
$X_0X_3X_5X_6$
$X_3X_5X_6X_7$
$X_0X_1X_3X_5$
$X_0X_1X_4X_5$
$X_1X_2X_3X_7$
$X_1X_2X_3X_7$

$F_4(i)$
$X_1X_2X_5X_6$
$X_1X_2X_5X_6$
$X_0X_3X_4X_7$
$X_1X_2X_5X_6$
$X_0X_3X_4X_7$
$X_1X_2X_5X_6$
$X_0X_3X_4X_7$
$X_0X_3X_4X_7$
$X_1X_2X_5X_6$
$X_0X_3X_4X_7$
$X_1X_2X_5X_6$
$X_1X_2X_5X_6$

$F_5(i)$
$X_1X_2X_3X_7$
$X_1X_2X_3X_7$
$X_2X_4X_6X_7$
$X_0X_4X_5X_6$
$X_0X_1X_3X_5$
$X_0X_4X_5X_6$
$X_2X_3X_6X_7$
$X_2X_4X_6X_7$
$X_3X_5X_6X_7$
$X_0X_3X_5X_6$
$X_1X_4X_5X_7$
$X_1X_4X_5X_7$

$F_6(i)$	$X_0X_1X_6X_7$
	$X_0X_2X_5X_7$
	$X_1X_2X_5X_6$
	$X_1X_3X_4X_6$
	$X_2X_3X_4X_5$
	$X_2X_3X_4X_5$
	$X_1X_3X_4X_6$
	$X_0X_2X_5X_7$
	$X_0X_2X_5X_7$
	$X_2X_3X_4X_5$
	$X_1X_3X_4X_6$
	$X_1X_3X_4X_6$

$F_7(i)$	$X_0X_1X_2X_3$
	$X_0X_1X_2X_3$
	$X_0X_1X_4X_5$
	$X_4X_5X_6X_7$
	$X_2X_3X_6X_7$
	$X_4X_5X_6X_7$
	$X_0X_1X_2X_4$
	$X_0X_1X_4X_5$
	$X_1X_2X_4X_7$
	$X_2X_4X_6X_7$
	$X_1X_3X_5X_7$
	$X_1X_3X_5X_7$

Table VI-2 : Unique terms of DCT

Input	Terms
8	$X_0X_1X_2X_3X_4X_5X_6X_7$
4	$X_0X_1X_2X_3$, $X_4X_5X_6X_7$, $X_0X_1X_2X_4$, $X_0X_1X_4X_5$, $X_0X_3X_5X_6$, $X_0X_1X_3X_5$, $X_0X_2X_4X_6$ $X_1X_2X_4X_7$, $X_2X_3X_4X_5$, $X_1X_3X_5X_7$, $X_0X_2X_5X_7$, $X_0X_1X_6X_7$, $X_1X_3X_4X_6$, $X_1X_2X_3X_7$ $X_3X_5X_6X_7$, $X_1X_4X_5X_7$, $X_0X_2X_3X_6$, $X_1X_2X_5X_6$, $X_0X_3X_4X_7$, $X_2X_4X_6X_7$, $X_2X_3X_6X_7$ $X_0X_4X_5X_6$

For the term with 8 inputs, three levels adder matrix is required. In the first level adder matrix, 4 two-input adders are used to generate 4. Eight input vectors are assigned to the adders stochastically. For the second level adder matrix, two part products with 4 input vectors will be obtained through 2 two-input adders with 4 outputs from the first level. Similar to the setting in the first level, 4 outputs are assigned randomly to the four input ports of the second level. In the last level, the third level, an adder will sum up the outputs from the second level and generate final result, the term with 8 inputs for DCT. Because of the final result containing all 8 inputs**,** the output of the third level can never be changed no matter how to change the configuration in the first and second levels.

For the 22 terms with 4 inputs, a two levels adder matrix can work, in which 22 adders are in the second level. The 44 inputs for the 22 adders are obtained from the outputs of the first level adder matrix. In theory, there are totally 28 ($=C_8^2$) different 2 inputs terms which can be implemented in the first level adder matrix. At the most twenty eight terms will be fed to the 44 input ports of the second level adder matrix. It means that these 2 inputs terms will be shared by two or more adders in the second level adder matrix. For example, the three adders generating terms $X_0X_1X_2X_4$, $X_0X_2X_4X_6$ and $X_1X_2X_4X_7$ respectively can share the 2 inputs term

X_2X_4. When only 20 or less 2-input terms are necessary to be implemented in the first level, it indicates that some 2 inputs terms are shared by more than two adders in the second level.

Obviously, the number of adders used in first level will directly determine the efficiency of the proposed processor. In the following paragraphs, the discussion will focus on the implementation of first level adder matrix with minimal adders .

B. Optimal Terms Sharing Scheme

The set R and input vectors will construct root and leaf level of a dimidiate tree respectively. Now, the 22 elements, as shown in Table VI-2, in set R and 8 inputs $\{X_0, X_1, \ldots, X_7\}$ build up all the nodes in their levels. Our purpose is to obtain the set $B_{best\_all}=\{B_0, B_1, \ldots, B_{p-1}\}$ which construct medium level of the dimidiate tree with the least elements.

Based on the discussion in section IV.5.4, a group of new subsets, R'_h, of R can be obtained as shown in Table VI-3, which contains all the sets in the root level with L_h element.

Table VI-3 : R'_h sets for DCT

L_h	R'_h
X_0	$X_0X_1X_2X_3$, $X_0X_1X_2X_4$, $X_0X_1X_4X_5$, $X_0X_3X_5X_6$, $X_0X_1X_3X_5$, $X_0X_2X_4X_6$, $X_0X_2X_5X_7$, $X_0X_1X_6X_7$, $X_0X_2X_3X_6$, $X_0X_3X_4X_7$, $X_0X_4X_5X_6$
X_1	$X_0X_1X_2X_3$, $X_0X_1X_2X_4$, $X_0X_1X_4X_5$, $X_0X_1X_3X_5$, $X_1X_2X_4X_7$, $X_1X_3X_5X_7$, $X_0X_1X_6X_7$, $X_1X_3X_4X_6$, $X_1X_2X_3X_7$, $X_1X_4X_5X_7$, $X_1X_2X_5X_6$
X_2	$X_0X_1X_2X_3$, $X_0X_1X_2X_4$, $X_0X_2X_4X_6$, $X_1X_2X_4X_7$, $X_2X_3X_4X_5$, $X_0X_2X_5X_7$, $X_1X_2X_3X_7$, $X_0X_2X_3X_6$, $X_1X_2X_5X_6$, $X_2X_4X_6X_7$, $X_2X_3X_6X_7$
X_3	$X_0X_1X_2X_3$, $X_0X_3X_5X_6$, $X_0X_1X_3X_5$, $X_2X_3X_4X_5$, $X_1X_3X_5X_7$, $X_1X_3X_4X_6$, $X_1X_2X_3X_7$, $X_3X_5X_6X_7$, $X_0X_2X_3X_6$, $X_0X_3X_4X_7$, $X_2X_3X_6X_7$
X_4	$X_4X_5X_6X_7$, $X_0X_1X_2X_4$, $X_0X_1X_4X_5$, $X_0X_2X_4X_6$, $X_1X_2X_4X_7$, $X_2X_3X_4X_5$, $X_1X_3X_4X_6$, $X_1X_4X_5X_7$, $X_0X_3X_4X_7$, $X_2X_4X_6X_7$, $X_0X_4X_5X_6$
X_5	$X_4X_5X_6X_7$, $X_0X_1X_4X_5$, $X_0X_3X_5X_6$, $X_0X_1X_3X_5$, $X_2X_3X_4X_5$, $X_1X_3X_5X_7$, $X_0X_2X_5X_7$, $X_3X_5X_6X_7$, $X_1X_4X_5X_7$, $X_1X_2X_5X_6$, $X_0X_4X_5X_6$
X_6	$X_4X_5X_6X_7$, $X_0X_3X_5X_6$, $X_0X_2X_4X_6$, $X_0X_1X_6X_7$, $X_1X_3X_4X_6$, $X_3X_5X_6X_7$, $X_0X_2X_3X_6$, $X_1X_2X_5X_6$, $X_2X_4X_6X_7$, $X_2X_3X_6X_7$, $X_0X_4X_5X_6$
X_7	$X_4X_5X_6X_7$, $X_1X_2X_4X_7$, $X_1X_3X_5X_7$, $X_0X_2X_5X_7$, $X_0X_1X_6X_7$, $X_1X_2X_3X_7$ $X_3X_5X_6X_7$, $X_1X_4X_5X_7$, $X_0X_3X_4X_7$, $X_2X_4X_6X_7$, $X_2X_3X_6X_7$

Based on these subsets in Table VI-3, a matrix of $P_{h,k}$ can be obtained as shown in Table VI-4, which contains leaf L_h ($h \in \{0, 1, \ldots, q-1\}$) appearance time in set R'_h.

Table VI-4 : First $P_{h,k}$ coefficient matrix for DCT

h \ k	0	1	2	3	4	5	6	7
0	0	5	5	5	5	5	5	*3*
1	5	0	5	5	5	5	*3*	5
2	5	5	0	5	5	*3*	5	5
3	5	5	5	0	*3*	5	5	5
4	5	5	5	3	0	5	5	5
5	5	5	3	5	5	0	5	5
6	5	3	5	5	5	5	0	5
7	*3*	5	5	5	5	5	5	0

The figures in Table VI-4 show that DCT is the special case for its coefficients are the same for most cases. But it does not affect the applying of the algorithm on it. First, we take 8 best $\{L_h, L_k \mid P_{h,k} = P_{h,max}\}$ pairs with the index of L_h ($h \in \{0, 1, \ldots, 7\}$), randomly selecting one if the appearance times are the same for each L_h. Eight pairs can be obtained, which are:

$$X_0X_6,\ X_1X_7,\ X_2X_4,\ X_3X_5,\ X_4X_2,\ X_5X_3,\ X_6X_0,\ X_7X_1$$

By eliminating duplicated pairs, we get the first 4 pairs as part of set B_{best_all} based on Table VI-4: X_0X_6, X_1X_7, X_2X_4 and X_3X_5. Subtracting these four pairs from 22 elements in set R, another 4 pairs are obtained which are also the elements of set B_{best_all}. They are X_0X_1, X_2X_3, X_4X_5 and X_6X_7. Apart from this second set of 4 pairs, there are still 8 elements in set R, which are left when first 4 pairs are subtracted from set R. They are :

$$X_0X_1X_2X_3,\ X_4X_5X_6X_7,\ X_0X_1X_4X_5,\ X_0X_2X_5X_7,X_1X_3X_4X_6,\ X_1X_2X_5X_6,\ X_0X_3X_4X_7,\ X_2X_3X_6X_7$$

When the second set of 4 pairs is subtracted from the rest 8 elements in set R, only 4 elements are left. They are:

$$X_0X_2X_5X_7,X_1X_3X_4X_6,\ X_1X_2X_5X_6,\ X_0X_3X_4X_77$$

After the first round processing, we have specified 8 elements in set B_{best_all} and 4 nodes are left which have not children nodes. It means that all the root nodes except 4 ones listed above can be presented as the summations by its 8 descendant nodes in medium level without order. The rest nodes in set B_{best_all} still need to be specified to complete the whole dimidiate tree.

By repeating the same steps, another $P_{h,k}$ coefficient matrix of the rest 4 nodes in set R is obtained, as shown in Table VI-5.

Table VI-5 : Second $P_{h,k}$ coefficient matrix for DCT

h \ k	0	1	2	3	4	5	6	7
0	0	0	1	1	1	1	0	*2*
1	0	**0**	1	1	1	1	*2*	0
2	1	1	0	0	0	*2*	1	1
3	1	1	0	0	*2*	0	1	1
4	1	1	0	*2*	0	0	1	1
5	1	1	*2*	0	0	**0**	1	1
6	0	*2*	1	1	1	1	0	0
7	*2*	0	1	1	1	1	0	0

In this table, the special situation found in Table VI-4 no longer happens. By selecting the maximal $P_{h,k}$ for each L_h, we obtain eight best pairs, which are:

$$X_0X_7,\ X_1X_6,\ X_2X_5,\ X_3X_4,\ X_4X_3,\ X_5X_2,\ X_6X_1,\ X_7X_0$$

If neglecting repeated ones, we will get 4 pairs which are X_0X_7, X_1X_6, X_2X_5 and X_3X_4. These four pairs can represent all 4 remaining root nodes as the summations of pairs.

By now, the work of searching for all nodes in set B_{best_all} for medium level is done. These nodes are the best sharing common terms for the proposed architecture in DCT implementation, which are:

$$X_0X_6,\ X_1X_7,\ X_2X_4,\ X_3X_5,\ X_0X_1,\ X_2X_3,\ X_4X_5,\ X_6X_7,\ X_0X_7,\ X_1X_6,\ X_2X_5,\ X_3X_4$$

By applying the algorithm for searching for the best set, 12 elements in set B_{best_all} are obtained. Compared with a universal set with 28 ($=C_8^2$) possible elements, the algorithm achieves 57% reduction in element number which will directly deduce the area and power consumption of hardware implementation.

It is noted that 12 elements in set B_{best_all} is a necessary condition for building the whole set R. But, for a single element, these 12 elements is a sufficient condition for implementation. Taking the element $X_0X_1X_6X_7$ in set R as an example, this element can be decomposed into $X_0X_6+ X_1X_7$, $X_0X_1+ X_6X_7$ or $X_0X_7+ X_1X_6$. Obviously, these six elements in set B_{best_all} are sufficient for implementing $X_0X_1X_6X_7$. Therefore, the rule for specifying parent-child relationship between nodes in two levels is that the times for adopting each node in medium level are as close as possible to average number**.** It can avoid occurrence of high output load capacitance on single nodes.

Considering the overall common terms occurrence, $X_0X_7+ X_1X_6$ is used in our design since these two node's fan out is less than the other nodes.

By sharing common terms, a total of 35 (12+22+1) two-input adders are needed for the DCT implementation, which gives 94.8% reduction in the number of adders compared with the 672 adders required by the theoretic implementation without optimization.

VI.1.6. Performance & Evaluation

With the common terms discussed in the previous sections, the 8 points 1-D and 2-D DCT was implemented with the proposed reconfigurable architecture.

A standard-cell based synthesis and layout was performed with Design Compiler from Synopsys, Inc., targeting the UMC 0.18 μm CMOS technology library. The power consumption was obtained by the Synopsys PrimePower. The area of the 2-D DCT is 1448062 μm^2 and the power consumption is 19.23mW at 20MHz system clock. The area of the 1-D DCT is 600929 μm^2 and the power consumption is 15.2mW at 20MHz system clock. The design can run with up to 144MHz (6.93ns) and 112-bits (=14bits$\times$8) outputs. This implies that our architecture can reach up to 16.128Gbps for the 1-D DCT. It can be

seen from power experimentation data that the power consumption of 2-D DCT is not twice the power consumption of 1-D DCT.

The reason is that the power consumption we used here for comparison is not the total consumed power for a group of data but the average consumed power in a time unit. The proposed implementation of 2-D DCT can be treated as using a 1-D DCT hardware circuit twice in two clock cycles for generating one 2-D DCT output. Therefore, from this point, the difference in power consumption between 2-D DCT and 1-D DCT is the power consumption of registers matrix which is not available for 1-D DCT. In the performance evaluation of this section, the comparison is focused on area, power and delay for 1-D DCT because the performance of proposed architecture can be fully revealed based on the target application.

A. Compared with the CSD Implementation

To compare with the performance of common subexpression elimination with CSD code, two implementations from [85, 86] are taken. All the implementations including ours are targeted at 8X8 DCT with bit width of 8. The number of required adders is 65 and 130 respectively in [85] and [86]. For our architecture, a total of 35 adders are needed in three levels to obtain all the products. This indicates our method achieves 46% and 73% reduction respectively compared with the implementations of existing CSD common subexpression elimination. The figures prove that adopted strategy is efficient and scheme selected is optimal.

B. Comparison between FPGA Device and Proposed Architecture

As similar domain-specific reconfigurable DA architecture cannot be found in the literature, an FPGA device is used as a reference to compare performance of our architecture in terms of area, power and delay. Based on the fact that the difference in voltage supply and CMOS technology will greatly affect chip power consumption and area, Xilinx Virtex-E [10] is taken as the reference FPGA device for implementing 1-D DCT, whose parameters are the same as the proposed architecture.

The area estimation of Virtex-E is based on two LUTs per slice where an estimated area of $3303\,\mu m^2$ is used per slice and its surrounding routing [73]. This value excludes the area of

memory which is embedded in FPGA device. The target FPGA device is larger than our architecture, so quiescent power consumed by unused computing units, RAM, clock manager and controller in FPGA device is very high. To remove all these impact factors, only the Vccint dynamic power consumption is taken as the reference. The delay time of FPGA device is the length from the rising edge of the clock to the time when signal reaches the output port, namely clock path plus data path.

The power consumption of 1-D DCT implementation on Virtex-E FPGA is 706mW with 1460 used slices which indicates 4822380 μm^2 areas are occupied. The delay of the implementation is 36.56ns which means the maximum frequency for 1-D DCT implementation on FPGA is 27.35MHz.

It is clear from the above experimentation data, our architecture achieves at least 97.8% reductions in power consumption compared with the FPGA Vccint dynamic power consumption with less than 87.5% area occupation. Our architecture can run more than 5 times faster than the FPGA implementation besides its merits in area and power consumption. Therefore, it can be concluded that our architecture achieves a good performance in terms of area, power consumption and speed.

It is noted that FPGA device is designed for general purpose and fits for any application if the target device scale is large enough. The architecture in this thesis is a domain-special one which is only available for DA applications. The comparison against FPGA device is used as a reference to evaluate the performance of our architecture. More comparison between ours and other ADIC designs will be made in the following sub-sections.

C. Power Consumption Comparison

To evaluate the power consumption of our architecture, an ASIC design in [127] is taken as an example, which adopts similar algorithm with ours. The power consumption in [127] is 12.45mW for 1D DCT with STMicroelectronics, hcmos9, 0.12μm technology at 1.5V, 50MHz. Considering dissipated power is approximately proportional to the square of supply voltage, the power consumption of the design in [127] can be scaled to 7.97mW for 1.2V. Our architecture consumes 7.13mW with UMC 0.13 μm CMOS technology library at 1.2V,

50MHz. It is noted that the power consumption of our architecture includes the power dissipation caused by interconnection network which brings the architecture powerful reconfigurability. The outstanding power characteristic of architecture makes it attractive to the power sensitive applications.

D. Area & Delay Comparison

To make the fair comparison, one of the key factors is to choose a proper reference design. To evaluate the area & delay performance of our architecture, we need to make comparisons with alternative solutions. However, currently there has been no existing architecture specifically designed for the distributed arithmetic applications. Therefore, only two implementations can be used for comparison: FPGA and ASIC implementation.

The proper comparison is hard to be made between FPGA, domain-specific architecture and ASIC. The comparison between different implementations should not only target at the realized function, but also at the potential ability they have. Selecting an FPGA implementation with the same function, implementation will make our architecture successive in area, power and delay comparison. However, it is meaningless.

It is clear that the reconfigurability is obtained at the cost of time, area, and power consumption. The more requirements are met, the higher the cost is. For a specific function, an ASIC implementation is the most efficient among all implementations, including FPGAs and domain-specific reconfigurable architectures. The general purpose FPGA devices, as well as digital signal processors, can be used for a wide range of applications. However, this powerful functionality leads to low efficiency for specific functions.

In this dissertation, to make the comparison with ASIC designs, we remove the reconfigurability from our architecture. The internal routing network in our design is used for re-arranging inner signals when the architecture switches to other applications, while there is not such a part in ASIC design. Given the fact that the internal routing network of proposed architecture consumes over 80% area of the whole architecture, the normalized delay-area product of our design in Table VI-6 is reduced by 80%. This scale can be regarded as the routing network which is replaced by fixed lines whose area and time

consumption can be ignored. The comparison between proposed reconfigurable architecture and ASIC intends to give an idea of area &delay performance of the proposed architecture.

As area can often be traded for delay and to eliminate the impact of different technologies, normalized delay-area product [128] is adopted to evaluate our architecture. It is defined as the product of the hardware cost (NAND gate count) and normalized average computation time which is the consumption time normalized by the delay of a NAND gate. This is used to evaluate the design performance in area and speed together. The lower the normalized delay-area product of a design, the better the performance of that design. The normalized delay-area products for different designs are listed in Table VI-6.

Therefore, several ASIC solutions for 1D DCT with the same throughput will be taken as the reference for evaluating the performance. The performances of some existing designs with 12-bit word length of data path are listed in Table VI-6.

It can be concluded from the table that our architecture achieves better performance than the average of 6 selected reference designs.

Table VI-6: Performances of some existing designs and ours

Designs	**normalized delay-area product Index ($\times 10^6$)**
[65]	2.0
[128]	1.0
[129]	1.3
[130]	2.2
[131]	1.1
[132]	0.9
Average	1.4
Proposed (Scaled)	1.2

VI.1.7. Summary

Eight points 1-D and 2-D DCT are mapped onto the architecture for the functionality verification and performance evaluation. Based on dimidiate tree, crossing forest, algorithm for searching for optimal scheme of common term sharing and its implementation which are introduced and defined to efficiently mapping and fully used hardware, the adder-based DA can achieve 94.8% reduction in area in the case of a DCT implementation. Compared with the common subexpression elimination with CSD code, up to 73% saving is obtained in hardware resources. The results of the proposed architecture prove its efficiency in terms of area, power and speed.

In comparison with FPGA DCT implementation, our architecture achieves at least 97.8% reductions in Vccint dynamic power consumption with less than 87.5% area occupation. Our architecture can run more than 5 times faster than the FPGA implementation besides its merits in area and power consumption.

In comparison with the existing ASIC designs, the experimental data show that the proposed architecture achieves better performance in area and speed than the average of six selected ASIC designs when the impact of interconnection resource in our architecture is removed. The right policy for trading off area and speed makes the architecture even consume less power than the ASIC designs using a similar algorithm.

It can be concluded from our results that the proposed reconfigurable architecture could provide an efficient hardware platform for implementing in DCT application.

VI.2. DFT Implementation

In the field of digital signal processing, the discrete Fourier transform (DFT) plays an important role in the analysis, design, and implementation of discrete-time signal-processing algorithms and systems [129, 130]. DFT is one of the most important algorithms in mathematical, numerical, scientific, engineering, and technical applications. Some of the applications of the DFT algorithm include time series, wave analysis, and convolution, solving, linear differential equations, particle simulations, Poisson's equation solver and

digital signal processing [131, 132]. The Fourier transform, in general, is a central component in many signal analysis systems.

The DFT, with a transform length equal to a power of 2, is usually implemented with the fast Fourier transform (FFT). The fast Fourier transform (FFT) is widely used in signal processing and communication such as digital filtering, spectral analysis, and polyphase filter multicarrier demultiplexing (MCD) [133]. The main reason for its widespread use is the existence of efficient techniques for its computation. Furthermore, in modern genetics and biology, the FFT is extensively applied in biological sequence analysis [134].

Due to the popularity of the orthogonal frequency division multiplex (OFDM) system, the demand for high-speed and low-power FFT emerges from various applications. The combination of the multiple-input multiple-output (MIMO) signal processing with OFDM communication system is considered as a promising solution to enhancing the data rates of the wireless communication systems of next generation operating in frequency-selective fading environments. The High Throughput Task Group which establishes IEEE 802.11n standard is going to draw up the next-generation wireless local area network (WLAN) proposal to deliver higher bandwidth based on the 802.11 a/g which is the current OFDM-based WLAN standards [135]. The fourth-generation cellular phone and the forthcoming new WLAN systems may also incorporate OFDM system to deliver higher bandwidth [136].

The FFT is one of the most critical components in OFDM systems. It directly affects the accuracy of the channel estimation as well as the symbol demapper. As the data transmission rate of OFDM systems increases, generating OFDM symbols with high data rate requires very high speed FFT processor. According to the European digital video/audio broadcasting (DVB-T/DAB) standards, an OFDM system may require FFT length ranging from 256 to 8192 points. Wireless local area network (WLAN) and HIPERLAN/2 systems require high-speed and low-power FFT/IFFT design [137, 138].

With the introduction of the radix-2 FFT by Cooley–Tukey in 1965 [139], considerable research has been carried out resulting in a number of algorithms. The FFT algorithms are based on the principle of decomposing the computation of DFT into sequences of smaller

DFTs. The first efficient FFT algorithm was discovered by Gauss in the 18th century and rediscovered by Cooley and Tukey [139] in 1960s. Later advances in the research of FFT algorithms include the higher radix FFT [140], the mixed-radix FFT [141], the prime-factor FFT [142], Winograd Fourier Transform Algorithm (WFTA) FFT [143], the split-radix FFT [144], the recursive FFT [145], and the combination of decimation-in-time (DIT) and decimation-in-frequency (DIF) FFT algorithms [146]. Two widely used approaches are the fixed radix of Cooley–Tukey and the split radix, since they provide algorithms with regular computational structures. Most of these algorithms illustrate FFT with similar FFT diagrams, which evolved from the recursive nature of the FFT algorithms and are constructed by basic butterfly structure.

VI.2.1. DFT Algorithm

The N-point DFT performs the transformation of N-point time domain data into N-point frequency domain data. Discrete means that the data is sampled at given time instead of being continuous. The DFT operates on an N-point sequence of numbers $x(n)$, which is obtained through uniform sampling of a finite period of a continuous function. The DFT of $x(n)$ is written as $X(k)$, and is defined by equation (VI-7) [147].

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \qquad k = 0,\ 1,\ ...,\ N-1 \qquad \textbf{(VI-7)}$$

where W_N is defined as

$$W_N = e^{-j2\pi/N} = \cos(2\pi / N) - j\sin(2\pi / N) \qquad \textbf{(VI-8)}$$

The W_N^K is called twiddle factor which is a periodic function in the period N. In this dissertation, twiddle factors are also named as coefficients. It is clear in equation (VI-7) that, for each k, N complex multiplications and N-1 complex additions are needed to calculate $X(k)$. Hence, roughly $2N^2$ complex operations are required for the computation of a N-point DFT. Similar to DFT, inverse DFT can be given as follows [147].

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk} \qquad k = 0,\ 1,\ ...,\ N-1 \qquad \textbf{(VI-9)}$$

VI.2.2. FFT Algorithm

A. Basic FFT Algorithm

Fast Fourier Transform is a collection of algorithms to speed up the DFT by reducing the number of operations required. It was popularized by Cooley and Tukey in the 1960s [139]. Actually, more than a century earlier, a German mathematician Karl Gauss had used this method [148]. For the sake of simplicity, N is assumed to be a power of 2, meaning $N = 2^m$, where the power m is a positive integer. N-point input sequence $x(n)$ can be separated into two subsequences of length N=2. One subsequence consists of even components of $x(n)$, the other is composed of odd components. Therefore, equation (VI-7) can be deduced as follows [149].

$$X(k) = \sum_{n_{even}=0}^{N-2} x(n)W_N^{nk} + \sum_{n_{odd}=1}^{N-1} x(n)W_N^{nk} \qquad \textbf{(VI-10)}$$

If n in the even and odd summations are replaced by $2m$ and $2m$+1, respectively, equation (VI-10) can be written below.

$$X(k) = \sum_{m=0}^{N/2-1} x(2m)(W_N^2)^{mk} + \sum_{m=0}^{N/2-1} x(2m+1)(W_N^2)^{mk}W_N^k \qquad \textbf{(VI-11)}$$

However, it is easy to prove that $W_N^2 = W_N/2$, so,

$$X(k) = \sum_{m=0}^{N/2-1} x(2m)W_{N/2}^{mk} + W_N^k \sum_{m=0}^{N/2-1} x(2m+1)W_{N/2}^{mk} \qquad \textbf{(VI-12)}$$

$$= \sum_{m=0}^{N/2-1} x_{even}(m)W_{N/2}^{mk} + W_N^k \sum_{m=0}^{N/2-1} x_{odd}(m)W_{N/2}^{mk} \qquad \textbf{(VI-13)}$$

where subsequence $x_{even}(m)$ consists of the even-indexed components of $x(n)$, and subsequence $x_{odd}(m)$ consists of the odd-indexed components of $x(n)$. Due to the periodicity of W_N ($W^k_N = W_N^{k+lN}$), N/2-point DFTs of $x_{even}(m)$ and $x_{odd}(m)$ can be computed for only N=2 of the N values of k. Therefore, it leads to a reduction from N^2 to $N^2/2+N/2$ in the number of complex multiplications. For large N, about 50% multiplication operation savings can be achieved, compared to the direct calculation of the DFT by equation (VI-7).

B. Radix-2 FFT Algorithm:

Since N is a power of 2, if $N > 2$, the number of components of $x_{even}(m)$ and $x_{odd}(m)$ should also be even. Hence, they can also be separated further into subsequences consisting of their own even and odd components. However, $x_{even}(m)$ and $x_{odd}(m)$ are calculated from N/4-point DFTs. Repeat this decimation procedure for $log_2(N)$-1 times until sequences with only two components are gained in the last stage. A total of $log_2(N)$ stages can be produced by applying this decimation procedure. Each stage has $N/2$ complex multiplications by some power of W_N. The final stage is reduced to 2-point DFTs where no multiplications are required, since the twiddle factors are trivial numbers there. In each stage, DFTs from previous stage are broken into two smaller DFTs, and the preceding FFT is called radix-2 FFT. The input sequence (time sequence) is divided into two smaller sequences at each stage, hence the radix-2 FFT algorithm is called decimation-in-time (DIT) algorithm. Figure VI-4 shows the dataflow graph of an 8-point radix-2 DIT FFT. As can be seen in this figure, N=2 (here $N/2$ is 4) multiplications are required in each stage. Hence, a total number of only $N/2*log_2(N)$ complex multiplications are needed for computing an N-point FFT.

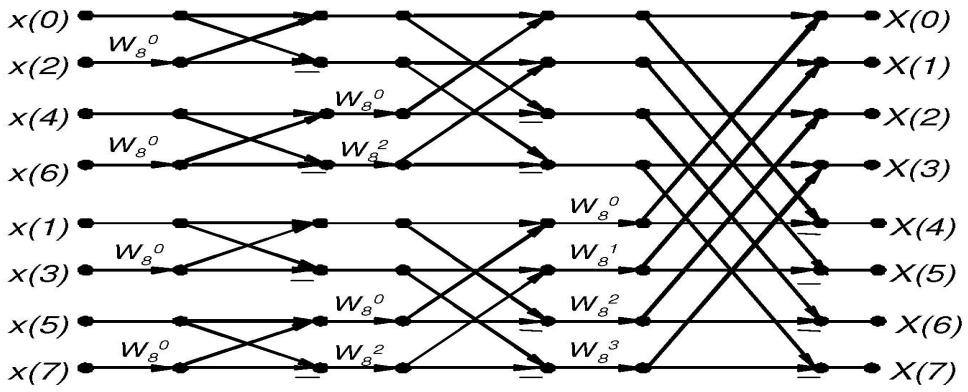

Figure VI-4 : Data flow graph of an 8-point radix-2 decimation-in-time FFT

C. Radix-4 and Mixed Radix FFT Algorithm

Radix-4 algorithm is more efficient than radix-2 algorithm, owing to the reduced stages and reduced number of cascaded multiplications, presumably leading to a more accurate result at the expense of additional computation. The radix-4 implementation only requires 1 stage versus 2 stages for a radix-2 implementation. It is suitable for N-point DFTs, where N is a

power of 4. Workload for a 4096 point FFT using different radices can be found in Table VI-7

Table VI-7 : Workload for a 4096 point FFT using different radices

Operation	radix-2	radix-4	radix-8
Complex multiplications	22528	15360	10752
Real multiplications	0	0	8192
Complex additions	49152	49152	49152
Real additions	0	0	8192
Memory accesses	49152	24576	16384

The development of radix-4 decimation-in-time FFT is similar to the development of radix-2 decimation-in-time FFT. The difference is that N-point input sequence $x(n)$ is split into four subsequences, $x(4n)$, $x(4n+1)$, $x(4n+2)$ and $x(4n+3)$ in a radix-4 decimation-in-time FFT. This decimation is recursive, until the final stage is implemented with 4-point DFTs. There isn't much difference between the Radix-8 and Radix-4 algorithms, except that the series split is N/8 instead of N/4. This brings with it the implicit difference in number of inputs processed in a single butterfly, the addressing of twiddle factors, number of stages being $\log_8(N)$.

The equation (VI-7) can be re-written by breaking the N-point DFT formula into four smaller DFTs as shown in equation (VI-14).

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{kn} \\ &= \sum_{n=0}^{N/4-1} x(n) W_N^{kn} + \sum_{n=N/4}^{N/2-1} x(n) W_N^{kn} + \sum_{n=N/2}^{3N/4-1} x(n) W_N^{kn} + \sum_{n=3N/4}^{N-1} x(n) W_N^{kn} \\ &= \sum_{n=0}^{N/4-1} x(n) W_N^{kn} + W_N^{Nk/4} \sum_{n=0}^{N/4-1} x\left(n+\frac{N}{4}\right) W_N^{kn} + \\ & W_N^{Nk/2} \sum_{n=0}^{N/4-1} x\left(n+\frac{N}{2}\right) W_N^{kn} + W_N^{3Nk/4} \sum_{n=0}^{N/4-1} x\left(n+\frac{3N}{4}\right) W_N^{kn} \end{aligned} \qquad \textbf{(VI-14)}$$

Due to the definition and periodicity of W_N, we have

$$W_N^{kN/4} = (-j)^k, \qquad W_N^{kN/2} = (-1)^k, \qquad W_N^{3kN/4} = (j)^k \tag{VI-15}$$

Thus, equation (VI-14) can be written as

$$X(k) = \sum_{n=0}^{N/4-1}\left[x(n) + (-j)^k x\left(n+\frac{N}{4}\right) + (-1)^k x\left(n+\frac{N}{2}\right) + (j)^k x\left(n+\frac{3N}{4}\right)\right] W_N^{nk} \tag{VI-16}$$

The equation (VI-16) is not an radix-4 FFT because the twiddle factor is dependent on N but not N/4. To convert it into radix-4 FFT, we subdivide the equation into four N/4-point subsequences, X(4k), X(4k+1), X(4k+2), and X(4k+3), k = 0, 1, ..., N/4. Thus we obtain the radix-4 FFT as

$$\begin{aligned}
X(4k) &= \sum_{n=0}^{N/4-1}\left[x(n) + x\left(n+\frac{N}{4}\right) + x\left(n+\frac{N}{2}\right) + x\left(n+\frac{3N}{4}\right)\right] W_N^{0} W_{N/4}^{kn} \\
X(4k+1) &= \sum_{n=0}^{N/4-1}\left[x(n) - jx\left(n+\frac{N}{4}\right) - x\left(n+\frac{N}{2}\right) + jx\left(n+\frac{3N}{4}\right)\right] W_N^{n} W_{N/4}^{kn} \\
X(4k+2) &= \sum_{n=0}^{N/4-1}\left[x(n) - x\left(n+\frac{N}{4}\right) + x\left(n+\frac{N}{2}\right) - x\left(n+\frac{3N}{4}\right)\right] W_N^{2n} W_{N/4}^{kn} \\
X(4k+3) &= \sum_{n=0}^{N/4-1}\left[x(n) + jx\left(n+\frac{N}{4}\right) - x\left(n+\frac{N}{2}\right) - jx\left(n+\frac{3N}{4}\right)\right] W_N^{3n} W_{N/4}^{kn}
\end{aligned} \tag{VI-17}$$

A 16-point, radix-4 decimation-in-frequency FFT algorithm is shown in Figure VI-5. Its input is in normal order and its output is in digit-reversed order.

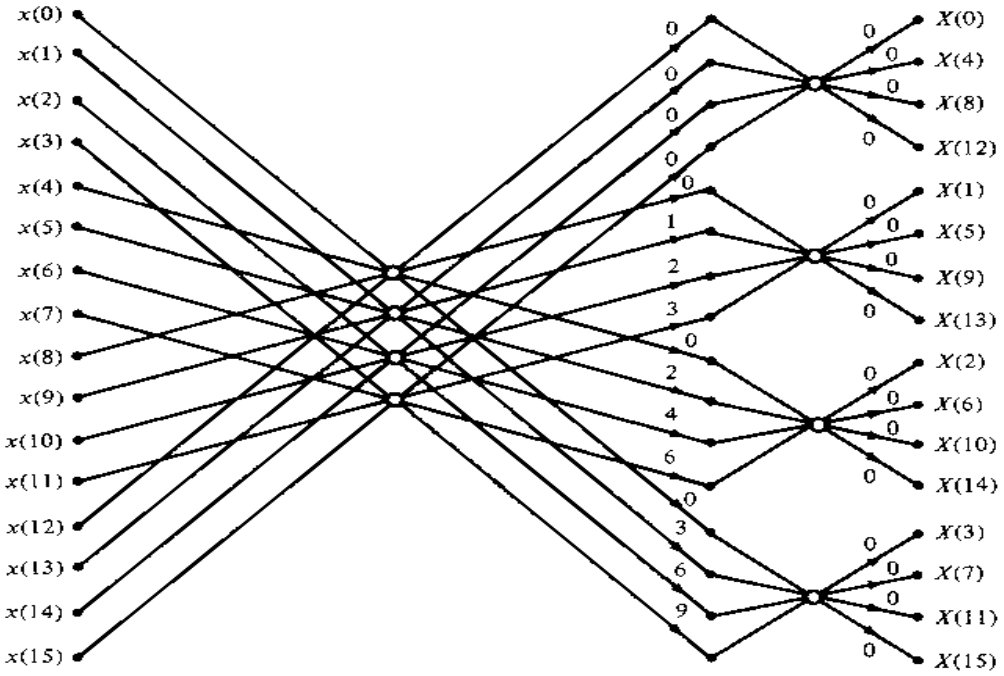

Figure VI-5 : Data flow graph of 16-point, radix-4 decimation-in-frequency FFT

For radix-r algorithms, such as radix-2 or radix-4, the butterfly elements used in each stage are the same. However, FFT algorithms, where the butterfly elements used in each stage are not all equal, are called mixed-radix algorithms [147]. For example, the butterflies in some stages are based on radix-2 algorithm; others are based on radix-4 algorithm or higher radices. Basically, radix-r algorithms excel mixed-radix algorithms, due to the consistency of butterflies in radix-r algorithms. However, through mixed-radix algorithms, the advantage of high radices can be applied to these conditions where N, the size of FFT, is not a power of the high radices. The examples, mixed-radix algorithms, are assigned for different FFT lengths as shown in Table VI-8, in which the higher radix is chosen first.

Table VI-8 : Mixed-radix algorithms for different FFT sizes

FFT size	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6
16					4	4
32				4	4	2
64				4	4	4
128			4	4	4	2
256			4	4	4	4
512		4	4	4	4	2
1024		4	4	4	4	4
2048	4	4	4	4	4	2
4096	4	4	4	4	4	4

VI.2.3. Overview of FFT Implementation

For various points FFT applications, radix-4 or mixed radix FFT algorithms can be adopted according to the delay, area and other performance requirements. The details for algorithms were discussed in section VI.2.2. For these FFT applications, the proposed processor can be used as a processing element (PE) and realize the function cooperating with other function units and control block. The FFT applications implementation is not the concern of this dissertation. In the following sub-sections, so we will focus on a 4-point FFT implementation on proposed processor.

As shown in equation (VI-8), the result of FFT algorithm contains two parts, real and imaginary part, which is different from DCT algorithm whose result has only one part. This characteristic of FFT indicates that the algorithm logic unit with 8 output ports in proposed

processor can process 4 input signals in time domain and generate 4 output results in frequency domain which contain 8 practical values.

In 4-point FFT algorithm implementation, the path of data flow is quite simple compared with DCT implementation. The input signals are routed to algorithm logic unit input ports through input matrix. The results from algorithm logic unit are put forward to output matrix and then output for further processing. It can be seen from FFT implementation that the data flow is one-way without data routing back and the results can be exported directly without temporary buffering.

The algorithm logic unit is divided into two function parts: real and imaginary part; The lower half part is used to generate the real part values of 4 final results and the rest half part, higher half part, is used to create 4 imaginary part values of results. Therefore, 4 input signals are replicated in input matrix for high 4 input ports of algorithm logic unit.

Because all 8 values of 4 final results of 4-point FFT are obtained from output ports of algorithm logic unit, registers matrix and its control unit are bypassed in FFT implementation. Output matrix takes the results and then routes them to corresponding output ports. It is noted that this configuration of registers matrix does not contradict with the requirements when the matrix is used for storing temporary data in high points FFT applications. The purpose of current configuration is to verify the function of proposed processor and obtain the performance in FFT implementation for evaluation.

VI.2.4. Algorithm logic Unit Implementation

The precision of FFT implementation lies in the coefficient representation when the input vector is given with the fixed precision. The bit-width of the twiddle factors is set to be 12 bits and the longer word-length is not cost efficient as the signal-to-quantization-noise ratio (SQNR) performance does not increase notably while the cost of multipliers and tables increases significantly [150]. The 12 bits word length in both real and imaginary parts for the proposed FFT implementation will also meet IEEE 802.11 standard requirements [135].

Following the steps of adder-based DA, equation (VI-8) can be presented in 2's complement format as shown in Figure VI-6, in which, the 4 values in real part of twiddle

factors are expressed with symbols $\mathrm{Re}W^{0\cdot k}{}_4$, $\mathrm{Re}W^{1\cdot k}{}_4$, $\mathrm{Re}W^{2\cdot k}{}_4$ and $\mathrm{Re}W^{3\cdot k}{}_4$ and 4 values in imaginary part of twiddle factors are expressed with symbols $\mathrm{Im}W^{0\cdot k}{}_4$, $\mathrm{Im}W^{1\cdot k}{}_4$, $\mathrm{Im}W^{2\cdot k}{}_4$ and $\mathrm{Im}W^{3\cdot k}{}_4$.

Due to lots of zeros in twiddle factors as shown in Figure VI-6, there are a few terms left after deducting the zero and duplicate terms which need no further calculation. Take twiddle factors $\mathrm{Im}W^{0\cdot k}{}_4$ and $\mathrm{Im}W^{2\cdot k}{}_4$ as an example, all factors are zero. It means that no operation is required when input data are multiplied by the coefficient. Finally, only one term of 4 inputs, 2 terms of 2 inputs and 4 input data are required for Wallace tree multiplier matrix. They are $X_0X_1X_2X_3$, X_0X_2, X_1X_3, X_0, X_1, X_2 and X_3.

Compared with the complex case, 22 terms of 4 inputs in DCT application, common terms sharing scheme is really a simple one. In theory, there are 66 (= 3 × 22) possible common terms for 22 terms with 4 inputs if duplications are not taken into account. For the term $X_0X_1X_2X_3$ in 4-point FFT implementation, there are only three possible schemes which are $X_0X_1 + X_2X_3$, $X_0X_2 + X_1X_3$ and $X_0X_3 + X_1X_2$. Considering that the two 2-input terms, X_0X_2 and X_1X_3, are obligatory, it is natural to determine the best scheme which is $X_0X_2 + X_1X_3$.

According to the selected scheme and the terms to be output to Wallace tree multiplier matrix, totally 7 terms are easily implemented with two levels adder arrays. (We consider 4 input data as terms because they are also the results output from adder arrays.) Four input data are output through the bypass paths in two levels adder array. Two adders in the first level adder array are used to implement the terms X_0X_2 and X_1X_3. The two outputs of these adders go straight to Wallace tree multiplier matrix through the bypass path in the second level adder array. Meanwhile, the outputs are also routed to an adder in the second level to generate term $X_0X_1X_2X_3$.

By now, 7 terms are implemented with two levels adder arrays for 4-point FFT. Only 3 adders are used, which indicates that only a little power will be consumed by adder arrays. In the following sub-section, the performance of 4-point FFT implementation will be scaled and evaluated.

$$\mathrm{Re}\,W_4^{0\cdot k} = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{vmatrix} \quad \mathrm{Re}\,W_4^{1\cdot k} = \begin{vmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{vmatrix} \quad \mathrm{Re}\,W_4^{2\cdot k} = \begin{vmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{vmatrix}$$

$$\mathrm{Re}\,W_4^{3\cdot k} = \begin{vmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{vmatrix} \quad \mathrm{Im}\,W_4^{0\cdot k} = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix} \quad \mathrm{Im}\,W_4^{1\cdot k} = \begin{vmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$\mathrm{Im}\,W_4^{2\cdot k} = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix} \quad \mathrm{Im}\,W_4^{3\cdot k} = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{vmatrix}$$

Figure VI-6 : Twiddle factors of 4-point FFT

VI.2.5. Performance & Evaluation

Following the details of implementation discussed in the previous sections, the 4 points FFT was implemented with the proposed reconfigurable architecture.

A standard-cell based synthesis and layout was performed with Design Compiler from Synopsys, Inc., employing the UMC 0.18 μm CMOS technology library. The power consumption was obtained with the Synopsys PrimePower. The area of the 4-point FFT is 527529 μm^2 and the power consumption is 10.1mW at 20MHz system clock. The design can run with up to 144MHz (6.93ns) and 112-bits (=14bits$\times$8) outputs. This implies that our architecture can reach up to 16.128Gbps for the 4 points FFT which is the same as 1-D DCT.

Because there is not such an application adopting FFT with only 4 points, the number of points of most FFT applications ranks from 64 to 4k or even 8k, and the performance data of 4 points FFT is hard to find in ASIC implementation, domain-specific FPGAs, DSP embedded system, programmable processors or even reconfigurable FFT architecture. On the other hand, the implementation of high points FFT application is not the concern of this dissertation. The implementation of 4 points FFT, one kind of DA applications, with proposed processor is used to verify the functionality of the architecture.

As similar domain-specific reconfigurable DA architecture cannot be found in the literature, an FPGA device is used as a reference to compare performance in area, power and delay with our architecture. Xilinx Virtex-E [10] is taken as the reference FPGA device for implementing 4 points FFT, which is exactly the same as the one used in 1-D DCT evaluation and comparison. The details of this device can be found in section VI.1.6, in terms of voltage supply, CMOS technology, area estimation, running platform, power consumption data selection and so on.

The power consumption of 4 points FFT implementation on Virtex-E FPGA is 274mW with 902 used slices which indicates that 2979306 μm^2 area is occupied. The delay of the implementation is 36.56ns which means the maximum frequency for 4 points FFT implementation on FPGA is 27.35MHz.

It is clear from the above experimental data that our architecture achieves at least 96.3% reductions in power consumption compared with the FPGA Vccint dynamic power consumption with less than 82.3% area occupation. Our architecture can run more than 5 times faster than the FPGA implementation except its merits in area and power consumption. Therefore, it can be concluded that the proposed processor achieves a good performance in terms of area, power consumption and speed compared with FPGA device in 4 points FFT implementation.

As we have discussed in previous sections, FPGA contains a large amount of routing resource which is redundant for FFT implementation but necessary for implementing other applications. The proposed processor achieves a good performance in FFT implementation when compared with FPGA device. But it is not the most efficient one in various reconfigurable architectures which are designed specially for FFT applications [151-153]. Actually, our processor is superior to FPGA device in the same way that these architectures achieve better performance than our processor in FFT domain, that is, the routing resource in proposed processor is redundant for FFT implementation but necessary for implementing other DA applications such as DCT.

VI.3. Conclusion

In this chapter, two widely-adopted DA applications, DCT and DFT, are implemented with our architecture for the functionality verification and performance evaluation.

The definition and methods for implementations of two algorithms were introduced briefly at the beginning of each section. In DCT implementation, the configurations for control path, registers matrix and algorithm logic unit were discussed and specified according to the requirements of the application. Based on dimidiate tree, crossing forest, algorithm for searching for optimal scheme of common term sharing and its implementation which were introduced and defined for efficient mapping and full use of hardware, the common term sharing scheme was obtained by applying dimidiate tree and the algorithm for searching for optimal scheme.

The experiential data of DCT and FFT implementation show the validity of algorithm and efficiency and functionality of proposed processor.

Compared with FPGA implementation, our architecture achieves at least 97.8% reductions in power consumption, less than 87.5% area occupation and more than 5 times faster for DCT implementation and 96.3% reductions in power consumption and more than 82.3% area saving for FFT implementation. Additionally, in comparison with existing ASIC DCT designs, the proposed architecture achieves better performance in area and speed than the average of six selected ASIC designs when the impact of interconnection resource in our architecture is removed.

It can be concluded from our results that the proposed reconfigurable architecture can provide an efficient hardware platform for implementing DA application. The right policy for trading off area and speed, common terms sharing architecture and algorithm for optimal scheme make this platform implement DA applications flexibly at the low cost in terms of area, power and delay.

Chapter VII

Conclusion and Future Work

VII.1. Conclusion

In this dissertation, we have presented a novel reconfigurable low-power processor for DA, a specific domain. This domain specific reconfigurable processor features high efficiency in terms of area, power and delay. It is a hybrid between traditional ASICs and general reconfigurable architectures such as FPGA devices. The goal of the novel architecture is to get close to the performance of ASICs, while maintaining the flexibility of programmable platforms.

DA algorithms can be frequently found in a wide variety of real world algorithms, e.g. DCT, DFT and DWT, used in digital image/signal processing including compression and beam forming applications. Because of the complexity of these algorithms, which are computationally intensive for large size applications, the computing power they consumed is enormous. The processor presented in this dissertation can be used to implement complex, high performance DA algorithms for communication and image processing applications with low cost in area and power compared with the traditional methods.

The performance and efficiency of the proposed architecture have been demonstrated and validated in the preceding chapters through the implementation of DCT and DFT which are widely used in most still picture compression standards, video conferencing standards and communication standards such as IEEE 802.11n, 802.11 a/g and WiMax.

A simple reconfigurable low power control unit in the processor is implemented with good performance in area, power and timing. The generic characteristic of the architecture makes it applicable for any small and medium size finite state machines which can be used as

control units to implement complex system behaviour and can be found in almost all engineering disciplines.

Furthermore, to map target application efficiently with the proposed architecture, a new algorithm is introduced for searching for the best common sharing terms set, which keeps the area and power consumption of implementation at low level. Some new concepts such as dimidiate tree and crossing forest are introduced and defined initially. They are used to describe the algorithm for common sharing terms set searching. A software implementation of this algorithm is presented, which can be used not only for the proposed architecture in this dissertation but also for all the implementations with adder-based distributed arithmetic algorithm.

In addition, some low power design techniques are applied in the architecture, such as unsymmetrical design style including unsymmetrical interconnection arranging, unsymmetrical PTBs selection and unsymmetrical mapping of basic computing units. All these design techniques achieve extraordinary power consumption saving. It is believed that they can be extended to more low power designs and architectures.

VII.2. Evaluation of Results and Contributions

VII.2.1. Novel and Efficient Points of the Work

Eight points 1-D and 2-D DCT are mapped onto the architecture for the functionality verification and performance evaluation. Compared with the common subexpression elimination with CSD code, up to 73% saving is obtained in hardware resources. In comparison with FPGA DCT implementation, our architecture achieves at least 97.8% reductions in Vccint dynamic power consumption with less than 87.5% area occupation. Our architecture can run more than 5 times faster than the FPGA implementation except its merits in area and power consumption. In comparison with existing ASIC designs, the experimental data show that the proposed architecture achieves better performance in area and speed than the average of six selected ASIC designs when the impact of interconnection resource in our architecture is removed.

In the FFT implementation of 4 points, the proposed architecture achieves at least 96.3% reductions compared with the FPGA Vccint dynamic power consumption and less than 82.3% area occupation. Our architecture can run more than 5 times faster than the FPGA implementation except its merits in area and power consumption.

Regarding the reconfigurable control unit architecture, ten test cases from the widely adopted FSM benchmark set are implemented using both ours and a FPGA device. It is demonstrated that our architecture can achieve an average reduction of 82% in power consumption, a decrease of 44% in area occupation and 20% reduction in delay when implementing the same circuit on a commercial FPGA device.

VII.2.2. Limitations

This subsection specifies exactly the extent of the restriction with the proposed low power reconfigurable DA processor:

- The presented architecture is targeted at DA applications. This departure point limits the applications of the processor to DA field only. Compared with FPGA devices which are for general purpose and can be applied in any application or field regardless of the limitation on scale or size, the scope of application fields of our processor is greatly less than that of FPGA.

- In this dissertation, only three DA applications, 1D-DCT, 2D-DCT and FFT, are implemented with the proposed architecture. There are still other DA applications such as DWT, DHT and so on which are widely adopted in digital signal processing. The functionality of processor needs further verification and more performance data with more DA applications.

- The last, probably the most noticeable limitation of the presented processor is the manual routing and mapping when a target application is implemented. Currently, all sub-modules including control unit, two-level adder structure, Wallace tree multiplier matrix, interconnection network and so on are configured based on the manual placing and routing which can take full advantage of the novel design and achieve the best efficiency in terms of area, power and speed. But manual routing and mapping will become an impossible mission when the target application is large.

VII.3. Future Work

The work undertaken during this Ph.D. project has concentrated on the development of novel low power DA processor for multimedia and telecommunication applications. A set of objectives for further research include:

- Implement more DA applications such as DWT, DHT and so on to further verify the functionality of processor and obtain more performance data.

- A serial DA algorithm can be implemented on the architecture, which consumes less power than parallel one but takes longer time for processing. Serial algorithm can be used for extreme power sensitive application without speed requirement.

- To expand the reconfigurable control unit to meet the requirements of complex applications and to make it an independent architecture to extend its applied applications.

- Some modification might be made on the architecture to extend its applicability in more applications or fields.

- To apply dynamic reconfigurable technology to the processor. This can make the processor change its function when it is running. Dynamic reconfigurable technology will improve the hardware efficiency and in the mean time, it will make the architecture larger and more complex because of extra hardware components.

- To develop automatic routing and mapping algorithm and tools. The Electronic Design Automation (EDA) tools are necessary and critical part in semiconductor design flow, in which the complexity of chip designing makes the manual routing and mapping impossible. Therefore, the algorithm and software for application placing, routing and mapping is the key step to make the processor extend to other designs or architectures.

- To explore a more generic architecture with low power consumption which can be applied in more fields, not just limited to certain application domain

Appendix

Publications from this work

- **Z. Liu**, T. Arslan, A.T. Erdogan, "A Novel Reconfigurable Distributed Arithmetic Architecture and its Application in Multimedia", *15th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (FPGA 2007), Monterey, California, February 18-20, 2007

- **Z. Liu**, T. Arslan, A.T. Erdogan, "A Novel Reconfigurable Low Power Distributed Arithmetic Architecture for Multimedia Applications", *12th Asia and South Pacific Design Automation Conference* (ASP-DAC 2007), Pacifico Yokohama, Yokohama, Japan, January 23-26, 2007

- **Z. Liu**, T. Arslan, A.T. Erdogan, "An Embedded Low Power Reconfigurable Fabric for Finite State Machine Operations", *2006 IEEE International Symposium on Circuits and Systems* (ISCAS 2006), Kos, Greece, 21-24 May 2006.

- **Z. Liu**, Khawam, T. Arslan, A.T. Erdogan, "A Low Power Heterogenous Reconfigurable Architecture For Embedded Generic Finite State Machines", *IEEE International SOC Conference* (SOCC 2005), pp. 113-114, Washington, DC, USA, September 25-28, 2005

- **Z. Liu**, T. Arslan, S. Khawam, I. Lindsay "A High Performance Synthesisable Unsymmetrical Reconfigurable Fabric For Heterogeneous Finite State Machines", *Asia and South Pacific Design Automation Conference 2005* (ASP-DAC 2005), pp. 639-642, Vol. 1, Shanghai, China, January 18 - 21, 2005

References

[1] J. M. Rabaey, M. J. Ammer, J. L. da Silva, Jr., D. Patel, and S. Roundy, "PicoRadio supports ad hoc ultra-low power wireless networking," *Computer,* vol. 33, pp. 42-48, 2000.

[2] R. Hartenstein, "Trends in reconfigurable logic and reconfigurable computing," in *Electronics, Circuits and Systems, 2002. 9th International Conference on*, 2002, pp. 801-808 vol.2.

[3] W. Tuttlebee, *Software Defined Radio: Baseband Technology for 3G Handsets and Basestations*: John Wiley & Sons, February 2004.

[4] http://www.nvidia.com/page/home.html.

[5] "The Cost of Design." vol. 19: IEEE Computer Society Press, 2002, p. 136.

[6] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao, "The Chimaera reconfigurable functional unit," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 12, pp. 206-217, 2004.

[7] W. J. C. Melis, P. Y. K. Cheung, and W. Luk, "Image registration of real-time video data using the SONIC reconfigurable computer platform," in *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on*, 2002, pp. 3-12.

[8] H. Z. Marlene Wan, Varghese George, Martin Benes, Arthur Abnous, Vandana Prabhu and Jan Rabaey, "Design Methodology of a Low-Energy Reconfigurable Single-Chip DSP System," *The Journal of VLSI Signal Processing,* vol. 28, Numbers 1-2, pp. 47-61, May, 2001.

[9] J. M. Rabaey, "Reconfigurable processing: the solution to low-power programmable DSP," in *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, 1997, pp. 275-278 vol.1.

[10] Xilinx_Inc., "Virtex™-E 1.8 V Field Programmable Gate Arrays Production Product Specification," DS022-1 (v2.3) ed, July, 2002.

[11] Altera_Inc., *Stratix II Device Family Data Sheet*, ver 4.3 ed. San Jose, May 2007.

[12] Xilinx_Inc., "Virtex-5 Family Overview - Product Specification," DS100 (v5.0) ed, February, 2009.

[13] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. A. B. D. Burger, "Clock rate versus IPC: the end of the road for conventional microarchitectures," in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, 2000, pp. 248-259.

[14] C. Katherine and H. Scott, "Reconfigurable computing: a survey of systems and software." vol. 34: ACM Press, 2002, pp. 171-210.

[15] L. David, B. Vaughn, J. David, L. Andy, L. Chris, L. Paul, M. Sandy, M. Cameron, P. Bruce, P. Giles, R. Srinivas, W. Chris, C. Richard, and R. Jonathan, "The stratix routing and logic architecture," in *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays* Monterey, California, USA: ACM, 2003.

[16] Atmel_inc., "Datasheets for AT40K05/10/20/40AL ", 2818F–FPGA–07/06 ed, July, 2007.

[17] "reconfiguration time," http://www.fpga-faq.com/archives/51175.html.

[18] S. Sezer, J. Heron, R. Woods, R. A. T. R. Turner, and A. A. M. A. Marshall, "Fast partial reconfiguration for FCCMs," in *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on*, 1998, pp. 318-319.

[19] G. Varghese, "Low energy field-programmable gate array." vol. PhD: University of California, Berkeley, 2000.

[20] C. R. Rupp, M. Landguth, T. Garverick, E. A. G. E. Gomersall, H. A. H. H. Holt, J. M. A. A. J. M. Arnold, and M. A. G. M. Gokhale, "The NAPA adaptive processing architecture," in *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on*, 1998, pp. 28-37.

[21] J. M. Arnold, "An architecture simulator for National Semiconductor's adaptive processing architecture (NAPA)," in *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on*, 1998, pp. 271-272.

[22] M. B. Gokhale and J. M. Stone, "NAPA C: compiling for a hybrid RISC/FPGA architecture," in *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on*, 1998, pp. 126-135.

[23] J. R. H. a. J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," in *IEEE Symposium on FPGAs for Custom Computing Machines*, Los Alamitos, CA, 1997, pp. 12-21.

[24] T. J. Callahan, J. R. Hauser, and J. Wawrzynek, "The Garp architecture and C compiler," *Computer,* vol. 33, pp. 62-69, 2000.

[25] Garp: http://brass.cs.berkeley.edu/garp.html.

[26] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. A. K. J. P. Kao, "The Chimaera reconfigurable functional unit," in *FPGAs for Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on*, 1997, pp. 87-96.

[27] Chimaera: http://www.ee.washington.edu/faculty/hauck/chimaera.html.

[28] K. Eguro and S. Hauck, "Issues and approaches to coarse-grain reconfigurable architecture development," in *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on*, 2003, pp. 111-120.

[29] R. Hartenstein, "Coarse grain reconfigurable architectures," in *Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific*, 2001, pp. 564-569.

[30] H. Zhining and S. Malik, "Managing dynamic reconfiguration overhead in systems-on-a-chip design using reconfigurable datapaths and optimized interconnection networks," in *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, 2001, pp. 735-740.

[31] Pleiades: http://bwrc.eecs.berkeley.edu/research/Configurable_Architectures/.

[32] A. Abnous and J. Rabaey, "Ultra-low-power domain-specific multimedia processors," in *VLSI Signal Processing, IX, 1996., [Workshop on]*, 1996, pp. 461-470.

[33] A. Abnous, K. Seno, Y. Ichikawa, M. Wan, and J. M. Rabaey, "Evaluation of a Low-Power Reconfigurable DSP architecture," in *5 th Reconfigurable Architectures workshop (RAW 98)*, March, 1998, pp. 55-60.

[34] Rapid: http://www.cs.washington.edu/research/lis/rapid/.

[35] E. Carl, C. C. Darren, and F. Paul, "RaPiD - Reconfigurable Pipelined Datapath," in *Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*: Springer-Verlag, 1996.

[36] C. Ebeling, D. C. Cronquist, P. Franklin, J. A. S. J. Secosky, and S. G. A. B. S. G. Berg, "Mapping applications to the RaPiD configurable architecture," in *FPGAs for Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on*, 1997, pp. 106-115.

[37] D. C. Cronquist, C. Fisher, M. Figueroa, P. A. F. P. Franklin, and C. A. E. C. Ebeling, "Architecture design of reconfigurable pipelined datapaths," in *Advanced Research in VLSI, 1999. Proceedings. 20th Anniversary Conference on*, 1999, pp. 23-40.

[38] MorphoSys: http://www.eng.uci.edu/morphosys/.

[39] H. Singh, L. Guangming, L. Ming-Hau, E. A. F. E. Filho, R. A. M. R. Maestre, F. A. K. F. Kurdahi, and N. A. B. N. Bagherzadeh, "Morphosys: case study of a reconfigurable computing system targeting multimedia applications," in *Design Automation Conference, 2000. Proceedings 2000. 37th*, 2000, pp. 573-578.

[40] J. Davila, A. de Torres, J. M. Sanchez, M. A. S.-E. M. Sanchez-Elez, N. A. B. N. Bagherzadeh, and F. A. R. F. Rivera, "Design and implementation of a rendering algorithm in a SIMD reconfigurable architecture (MorphoSys)," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, 2006, p. 6 pp.

[41] H. Singh, L. Ming-Hau, L. Guangming, F. J. A. K. F. J. Kurdahi, N. A. B. N. Bagherzadeh, and E. M. C. A. F. E. M. C. Filho, "MorphoSys: a reconfigurable architecture for multimedia applications," in *Integrated Circuit Design, 1998. Proceedings. XI Brazilian Symposium on*, 1998, pp. 134-139.

[42] L. Guangming, H. Singh, L. Ming-Hau, N. A. B. N. Bagherzadeh, F. J. A. K. F. J. Kurdahi, E. M. C. A. F. E. M. C. Filho, and V. A. C.-A. V. Castro-Alves, "The MorphoSys dynamically reconfigurable system-on-chip," in *Evolvable Hardware, 1999. Proceedings of the First NASA/DoD Workshop on*, 1999, pp. 152-160.

[43] H. Singh, L. Ming-Hau, L. Guangming, F. J. A. K. F. J. Kurdahi, N. A. B. N. Bagherzadeh, and E. M. A. C. F. E. M. Chaves Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *Computers, IEEE Transactions on,* vol. 49, pp. 465-481, 2000.

[44] Chameleon: http://chameleon.ctit.utwente.nl/.

[45] G. Karypis, H. Eui-Hong, and V. Kumar, "Chameleon: hierarchical clustering using dynamic modeling," *Computer,* vol. 32, pp. 68-75, 1999.

[46] J. M. S. Gerard, B. Ties, J. M. H. Paul, J. M. Sape, and S. Jaap, "Chameleon - Reconfigurability in Hand-Held Multimedia Computers," in *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing* Karlsruhe, Germany: Springer-Verlag, 1999.

[47] A. A. Emira, A. Valdes-Garcia, X. Bo, A. N. A. M. A. N. Mohieldin, A. Y. A. V.-L. A. Y. Valero-Lopez, S. T. A. M. S. T. Moon, A. C. X. Chunyu Xin, and E. A. S.-S. E. Sanchez-Sinencio, "Chameleon: a dual-mode 802.11b/Bluetooth receiver system design," *Circuits and Systems I: Regular Papers, IEEE Transactions on [Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on],* vol. 53, pp. 992-1003, 2006.

[48] G. J. M. Smit, P. J. M. Havinga, M. Bos, L. T. Smit, and P. M. Heysters, "Reconfiguration in Mobile Multimedia Systems," in *1st PROGRESS workshop on Embedded Systems*, 2000, pp. 95-105.

[49] P. M. Heysters, H. Bouma, J. Smit, G. J. M. Smit, and P. J. M. Havinga, "Reconfigurable System Design: The Control Part," in *2nd PROGRESS workshop on Embedded Systems*, The Netherlands, 2001.

[50] Elixent_Limited, "The Reconfigurable Algorithm Processor," http://www.elixent.com/products/white-papers.htm.

[51] Elixent_Limited, "The Reconfigurable Algorithm Processor," http://www.elixent.com/assets/WP0001_D_Fabrix_Apps.pdf.

[52] T. Stansfield, "Using Multiplexers for Control and Data in D-Fabrix " in *13th International Conference on Field-Programmable Logic and Applications , FPL 2003*, Lisbon, Portugal, 2003, pp. 416-425.

[53] M. Alan, S. Tony, K. Igor, V. Jean, and H. Brad, "A reconfigurable arithmetic array for multimedia applications," in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, Monterey, California, United States, 1999, pp. 135-143.

[54] V. Betz, J. Rose, and A. Morquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Boston MA: Kluwer Academic Publishers, 1999.

[55] T. William, M. Kip, J. Atul, H. Randy, W. Norman, T. Tony, R. Omid, G. Varghese, W. John, Andr, and DeHon, "HSRA: high-speed, hierarchical synchronous reconfigurable array," in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays* Monterey, California, United States: ACM, 1999.

[56] S. A. White, "Applications of distributed arithmetic to digital signal processing: a tutorial review," *ASSP Magazine, IEEE [see also IEEE Signal Processing Magazine],* vol. 6, pp. 4-19, 1989.

[57] S. Zohar, "Fast Hardware Fourier Transformation Through Counting," *Transactions on Computers,* vol. C-22, pp. 433-441, 1973.

[58] S. Zohar, "The Counting Recursive Digital Filter," *Transactions on Computers,* vol. C-22, pp. 338-347, 1973.

[59] S. Zohar, "New Hardware Realizations of Nonrecursive Digital Filters," *Transactions on Computers,* vol. C-22, pp. 328-338, 1973.

[60] S. Zohar, "A Realization of the RAM Digital Filter," *Transactions on Computers,* vol. C-25, pp. 1048-1052, 1976.

[61] A. Peled and L. Bede, "A new approach to the realization of nonrecursive digital filters," *Audio and Electroacoustics, IEEE Transactions on,* vol. 21, pp. 477-484, 1973.

[62] A. Peled and L. Bede, "A new hardware realization of digital filters," *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on,* vol. 22, pp. 456-462, 1974.

[63] G. S. Stewart and B. D. Peter, "Serial-data computation," Kluwer Academic Publishers, 1988, p. 239.

[64] T. S. Chang, C. Chen, and C. W. Jen, "New distributed arithmetic algorithm and its application to IDCT," *Circuits, Devices and Systems, IEE Proceedings-,* vol. 146, pp. 159-163, 1999.

[65] K. Dae Won, K. Taek Won, S. Jung Min, Y. Jae Kun, L. Suk Kyu, S. Jung Hee, and C. Jun Rim, "A compatible DCT/IDCT architecture using hardwired distributed

arithmetic," in *The 2001 IEEE International Symposium on Circuits and Systems*, 2001, pp. 457-460 vol. 2.

[66] G. Jiun-In, "A new DA-based array for one dimensional discrete Hartley transform," in *The 2001 IEEE International Symposium on Circuits and Systems*, 2001, pp. 662-665 vol. 4.

[67] L. Zhenyu, S. Khawam, T. Arslan, and A. T. Erdogan, "A Low Power Heterogenous Reconfigurable Architecture For Embedded Generic Finite State Machines," in *IEEE International SOC Conference*, 2005, pp. 113-114.

[68] M. R. Boschetti, A. M. S. Adario, I. S. Silva, and S. Bampi, "Techniques and mechanisms for dynamic reconfiguration in an image processor," in *15th Symposium on Integrated Circuits and Systems Design*, 2002, pp. 177-182.

[69] M. P. Leong and P. H. W. Leong, "A variable-radix digit-serial design methodology and its application to the discrete cosine transform," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 11, pp. 90-104, 2003.

[70] J. G. Liu, H. F. Li, F. H. Y. Chan, and F. K. Lam, "Fast Discrete Cosine Transform via Computation of Moments," *J. VLSI Signal Process. Syst.,* vol. 19, pp. 257-268, 1998.

[71] A. K. Pai, K. Benkrid, and D. Crookes, "Embedded reconfigurable DCT architectures using adder-based distributed arithmetic," in *Seventh International Workshop on Computer Architecture for Machine Perception*, 2005, pp. 81-86.

[72] J. Park and K. Roy, "A low power reconfigurable DCT architecture to trade off image quality for computational complexity," in *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, 2004, pp. V-17-20 vol.5.

[73] S. Khawam, T. Arslan, and F. Westall, "Synthesizable reconfigurable array targeting distributed arithmetic for system-on-chip applications," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004, p. 150.

[74] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*: John Wiley & Sons, 1999.

[75] N. Pippenger, "On Crossbar Switching Networks," *Communications, IEEE Transactions on [legacy, pre - 1988],* vol. 23, pp. 646-659, 1975.

[76] Clos and Charles, *A study of non-blocking switching netwroks* vol. 32: Bell System Tech Journal, 1953.

[77] V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*: Academic Pr June 1965.

[78] G. H. Chapman and K. Fang, "Comparison of laser link crossbar and Omega network switching for wafer-scale integration defect avoidance," in *Wafer Scale Integration, 1994. Proceedings., Sixth Annual IEEE International Conference on*, 1994, pp. 352-361.

[79] L. Guy and L. David, "Using sparse crossbars within LUT," in *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays* Monterey, California, United States: ACM Press, 2001.

[80] A. Yavuz Oruc and H. M. Huang, "Crosspoint complexity of sparse crossbar concentrators," *Information Theory, IEEE Transactions on,* vol. 42, pp. 1466-1471, 1996.

[81] S. Nakamura and G. M. Masson, "Lower Bounds on Crosspoints in Concentrators," *Transactions on Computers,* vol. C-31, pp. 1173-1179, 1982.

[82] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova, "A new algorithm for elimination of common subexpressions," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 18, pp. 58-68, 1999.

[83] M. Martinez-Peiro, E. I. Boemo, and L. Wanhammar, "Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on],* vol. 49, pp. 196-203, 2002.

[84] O. Gustafsson and L. Wanhammar, "ILP modelling of the common subexpression sharing problem," in *9th International Conference on Electronics, Circuits and Systems*, 2002, pp. 1171-1174 vol.3.

[85] M. D. Macleod and A. G. Dempster, "Common subexpression elimination algorithm for low-cost multiplierless implementation of matrix multipliers," *Electronics Letters,* vol. 40, pp. 651-652, 2004.

[86] C. Tian-Sheuan, G. Jiun-In, and J. Chein-Wei, "Hardware-efficient DFT designs with cyclic convolution and subexpression sharing," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on],* vol. 47, pp. 886-892, 2000.

[87] K. Hwang, *Computer Arithmetic, Principles, Architecture, and Design*. New York: John Wiley & Sons, 1979.

[88] I. Koren, *Computer Arithmetic Algorithms*. Englewood Clis: Prentice-Hall, 1993.

[89] A. P. Vinod and E. M. K. Lai, "Hardware efficient DCT implementation for portable multimedia terminals using subexpression sharing," in *TENCON 2004. 2004 IEEE Region 10 Conference*, 2004, pp. 227-230 Vol. 1.

[90] M. Benhamid and M. Othman, "FPGA Implementation of a Canonical Signed Digit Multiplier-less based FFT Processor for Wireless Communication Applications," in *Semiconductor Electronics, 2006. ICSE '06. IEEE International Conference on*, 2006, pp. 641-645.

[91] S. He and M. Torkelson, "FPGA implementation of FIR filters using pipelined bit-serial canonical signed digit multipliers," in *Custom Integrated Circuits Conference, 1994., Proceedings of the IEEE 1994*, 1994, pp. 81-84.

[92] A. T. G. Fuller, B. Nowrouzian, and F. Ashrafzadeh, "Optimization of FIR digital filters over the canonical signed-digit coefficient space using genetic algorithms," in *Circuits and Systems, 1998. Proceedings. 1998 Midwest Symposium on*, 1998, pp. 456-459.

[93] R. M. Hewlitt and E. S. Swartzlantler, Jr., "Canonical signed digit representation for FIR digital filters," in *Signal Processing Systems, 2000. SiPS 2000. 2000 IEEE Workshop on*, 2000, pp. 416-426.

[94] M. L. Minsky, *Computation: Finite and Infinite Machines*: Prentice-Hall, 1967.

[95] W. L. Yang, R. M. Owens, and M. J. Irwin, "Multi-way FSM decomposition based on interconnect complexity," in *Design Automation Conference, 1993, with EURO-VHDL '93. Proceedings EURO-DAC '93. European*, 1993, pp. 390-395.

[96] J. C. Monteiro and A. L. Oliveira, "FSM decomposition by direct circuit manipulation applied to low power design," in *Design Automation Conference, 2000. Proceedings of the ASP-DAC 2000. Asia and South Pacific*, 2000, pp. 351-358.

[97] R. S. Shelar, H. Narayanan, and M. P. Desai, "Orthogonal partitioning and gated clock architecture for low power realization of FSMs," in *ASIC/SOC Conference, 2000. Proceedings. 13th Annual IEEE International*, 2000, pp. 266-270.

[98] J. Hartmanis, "Symbolic analysis of a decomposition of information processing," *Information Control,* vol. 3, pp. 154-178, June 1960.

[99] S. Devadas and A. R. Newton, "Decomposition and factorization of sequential finite state machines," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 8, pp. 1206-1217, 1989.

[100] M. Geiger and T. Muller-Wipperfurth, "FSM decomposition revisited: algebraic structure theory applied to MCNC benchmark FSMs," in *Design Automation Conference, 1991. 28th ACM/IEEE*, 1991, pp. 182-185.

[101] Xilinx_Inc., "CoolRunner XPLA3 CPLD Family Product Specification," DS012 (v2.3) ed, August 31, 2007.

[102] K. Noha, B. Kimberly, and J. E. W. Steven, "Architectures and algorithms for synthesizable embedded programmable logic cores," in *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays* Monterey, California, USA: ACM, 2003.

[103] A. Yan and S. J. E. Wilton, "Product-term based synthesizable embedded programmable logic cores," in *Field-Programmable Technology (FPT), 2003. Proceedings. 2003 IEEE International Conference on*, 2003, pp. 162-169.

[104] Benchmarks: http://www.cbl.ncsu.edu/benchmarks/LGSynth93.

[105] J. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: the effect of logic block functionality on area efficiency," *Solid-State Circuits, IEEE Journal of,* vol. 25, pp. 1217-1225, 1990.

[106] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays," *Proceedings of the IEEE,* vol. 81, pp. 1013-1029, 1993.

[107] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 12, pp. 288-298, 2004.

[108] L. Zhenyu, T. Arslan, S. Khawam, and I. Lindsay, "A high performance synthesisable unsymmetrical reconfigurable fabric for heterogeneous finite state machines," in *Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific*, 2005, pp. 639-644 Vol. 1.

[109] Xilinx_Inc., "XC2C128 CoolRunner-II CPLD Product Specification," DS093 (v2.9) ed, June, 2005.

[110] K. Kuusilinna, V. Lahtinen, T. Hamalainen, and J. Saarinen, "Finite state machine encoding for VHDL synthesis," *Computers and Digital Techniques, IEE Proceedings -,* vol. 148, pp. 23-30, 2001.

[111] N. Yevtushenko, S. Zharikova, and M. Vetrova, "Multi component digital circuit optimization by solving FSM equations," in *Digital System Design, 2003. Proceedings. Euromicro Symposium on*, 2003, pp. 62-68.

[112] Xilinx_Inc., "Virtex-II Platform FPGAs: Functional Description," DS031 (v3.4) ed, March, 2005.

[113] R. Koenen, *Overview of the MPEG-4 Standard* vol. N4668: ISO/IEC JTC1/SC29/WG11, March 2002

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

[115] "IEEE standard specifications for the implementations of 8x8 inverse discrete cosine transform," *IEEE Std 1180-1990,* 1991.

[116] C. Wen-Hsiung, C. Smith, and S. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform," *Communications, IEEE Transactions on [legacy, pre - 1988],* vol. 25, pp. 1004-1009, 1977.

[117] B. D. Tseng and W. C. Miller, "On Computing the Discrete Cosine Transform," *Transactions on Computers,* vol. C-27, pp. 966-968, 1978.

[118] F. A. Kamangar and K. R. Rao, "Fast Algorithms for the 2-D Discrete Cosine Transform," *Transactions on Computers,* vol. C-31, pp. 899-906, 1982.

[119] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete Cosine Transfom," *Transactions on Computers,* vol. C-23, pp. 90-93, 1974.

[120] W. Li, "A new algorithm to compute the DCT and its inverse," *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on],* vol. 39, pp. 1305-1313, 1991.

[121] C. Chakrabarti and J. Jaja, "Systolic architectures for the computation of the discrete Hartley and the discrete cosine transforms based on prime factor decomposition," *Transactions on Computers,* vol. 39, pp. 1359-1368, 1990.

[122] S. C. Chan and K. L. Ho, "Fast algorithms for computing the discrete cosine transform," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on],* vol. 39, pp. 185-190, 1992.

[123] K. J. R. Liu, C. T. Chiu, R. K. Kolagotla, and J. F. Jala, "Optimal unified architectures for the real-time computation of time-recursive discrete sinusoidal transforms," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 4, pp. 168-180, 1994.

[124] C. Nam Ik and L. San Uk, "Fast algorithm and implementation of 2-D discrete cosine transform," *Circuits and Systems, IEEE Transactions on,* vol. 38, pp. 297-305, 1991.

[125] P. Duhamel and C. Guillemot, "Polynomial transform computation of the 2-D DCT," in *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*, 1990, pp. 1515-1518 vol.3.

[126] H. R. Wu and F. J. Paoloni, "A two-dimensional fast cosine transform algorithm based on Hou's approach," *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on],* vol. 39, pp. 544-546, 1991.

[127] S. Ghosh, S. Venigalla, and M. Bayoumi, "Design and implementaion of a 2D-DCT architecture using architecture using coefficient distributed arithmetic," in *IEEE Computer Society Annual Symposium on VLSI*, 2005, pp. 162-166.

[128] G. Jiun-In, J. Rei-Chin, and C. Jia-Wei, "An efficient 2-D DCT/IDCT core design using cyclic convolution and adder-based realization," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 14, pp. 416-428, 2004.

[129] C. S. Burrus and T. W. Parks, *DFT/FFT and Convolution Algorithms and Implementation*. New York: Wiley, 1985.

[130] A. V. Oppenheim and C. M. Rader, *Discrete-Time Signal Processing*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 1999.

[131] E. O. Brigham, *The Fast Fourier Transform and Its Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[132] O. K. Ersoy, *Fourier-Related Transforms, Fast Algorithms and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1997.

[133] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[134] W. Wei and D. H. Johnson, "Computing linear transforms of symbolic signals," *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on],* vol. 50, pp. 628-634, 2002.

[135] S. A. Mujtaba, "TGn Sync Proposal Technical Specification." vol. 11-04-0889-05-000n: IEEE 802.11, May. 2005.

[136] M. Engels, W. Eberle, and B. Gyselinckx, "Design of a 100 Mbps wireless local area network," in *Signals, Systems, and Electronics, 1998. ISSSE 98. 1998 URSI International Symposium on*, 1998, pp. 253-256.

[137] N. Weste and D. J. Skellern, "VLSI for OFDM," *Communications Magazine, IEEE,* vol. 36, pp. 127-131, 1998.

[138] R. v. Nee and R. Prasad, *OFDM for Wireless Multimedia Communications*. Norwell, MA: Archtech House, 2000.

[139] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation,* vol. 19, April 1965.

[140] G. Bergland, "A radix-eight fast Fourier transform subroutine for real-valued series," *Audio and Electroacoustics, IEEE Transactions on,* vol. 17, pp. 138-144, 1969.

[141] R. Singleton, "An algorithm for computing the mixed radix fast Fourier transform," *Audio and Electroacoustics, IEEE Transactions on,* vol. 17, pp. 93-103, 1969.

[142] D. Kolba and T. Parks, "A prime factor FFT algorithm using high-speed convolution," *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on,* vol. 25, pp. 281-294, 1977.

[143] S. Winograd, "On computing the discrete Fourier transform," *Mathematics of Computation,* vol. 32 NO.141, pp. 175–199, Jan. 1978.

[144] P. Duhamel and H. Hollmann, "Split radix FFT algorithm," *Electronics Letters,* vol. 20, pp. 14-16, 1984.

[145] Z. Feihong, "Two-dimensional recursive fast Fourier transform [image processing applications," *Radar and Signal Processing, IEE Proceedings,* vol. 137, pp. 262-266, 1990.

[146] A. Saidi, "Decimation-in-time-frequency FFT algorithm," in *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, 1994, pp. III/453-III/456 vol.3.

[147] W. W. Smith and J. M. Smith, *Handbook of Real-time Fast Fourier Transforms*: Wiley-IEEE Press 2002.

[148] M. Heideman, D. Johnson, and C. Burrus, "Gauss and the history of the fast fourier transform," *ASSP Magazine, IEEE [see also IEEE Signal Processing Magazine],* vol. 1, pp. 14-21, 1984.

[149] N. Kalouptsidis, *Sigal Processing Systems: Theory and Design*: A Wiley-Interscience publication, 1997.

[150] S. Johansson, H. Shousheng, and P. Nilsson, "Wordlength optimization of a pipelined FFT processor," in *Circuits and Systems, 1999. 42nd Midwest Symposium on*, 1999, pp. 501-503 vol. 1.

[151] N. Kazuto, Y. Shingo, and M. Yoshikazu, "A study of dynamic reconfigurable FFT processor for OFDM based cognitive radio," in *Communications and Information Technologies, 2007. ISCIT '07. International Symposium on*, 2007, pp. 1507-1510.

[152] A. Ahmadinia, B. Ahmad, and T. Arslan, "System Level Modelling of Reconfigurable FFT Architecture for System-on-Chip Design," in *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*, 2007, pp. 169-175.

[153] Y. Zhao, A. T. Erdogan, and T. Arslan, "A low-power and domain-specific reconfigurable FFT fabric for system-on-chip applications," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005, p. 4 pp.