NANYANG TECHNOLOGICAL UNIVERSITY

Design and Optimization of DSP Architectures for Multi-Context FPGA with Dynamic Reconfiguration

Rakesh Vijayakumara Warrier

School of Computer Science and Engineering

A thesis submitted to Nanyang Technological University
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

December 2016

THESIS ABSTRACT

Design and Optimization of DSP Architectures for Multi-Context FPGA with Dynamic Reconfiguration

by

Rakesh Vijayakumara Warrier

Doctor of Philosophy

School of Computer Science and Engineering

Nanyang Technological University, Singapore

Field Programmable Gate Arrays (FPGAs) are now widely adopted as hardware accelerators due to their inherent parallel processing capability. However, the sub-optimal logic utilization and large reconfiguration latency in conventional single-context FPGAs pose constraints on their usage for applications like adaptive control systems in vehicles, software defined radio where frequent context switching or resource sharing between tasks are required. As such, multi-context FPGAs with dynamic reconfiguration capability have been introduced with the aim to allow rapid reconfiguration of the FPGA, and hence increase the effective logic density. The current generation of multi-context FPGAs typically use a dynamic reconfigurable architecture based on static Random Access Memory (RAM) to implement multiple configuration planes that enable fast switching between contexts. The main challenge of these types of multi-context FPGAs are limited on-chip storage and relatively long reconfiguration latency (of the order of milliseconds). With technology driving down to nano scale, new generations of hybrid multi-context FPGA architectures, such as the CMOS/NAnoTechnology reconfigURablE architecture (NATURE) that use on-chip nano RAMs to store multiple configurations to enable extremely fast runtime reconfiguration (of the order of pico seconds) have been developed. This type of FPGA enables cycle-by-cycle reconfiguration and temporal logic folding resulting in improved logic density and area delay product by more than an order of magnitude compared to traditional FPGAs. However, the fine granularity of this type of architecture limits its usage as a high performance hardware accelerator that implements compute intensive arithmetic operations.

This research work explores and presents how DSP architectures can be designed for the hybrid multi-context NATURE platform in order to fully exploit its advantages and possibilities. The performance of various compute intensive signal

processing kernels are used in the study to benchmark the improvements achieved by the proposed DSP architectures. A full-block dynamically reconfigurable DSP architecture is first presented, which can be reconfigured at runtime to implement different arithmetic functions in different clock cycles. To fully exploit the capability of temporal logic folding techniques in NATURE, the DSP block is then extended to support pipeline level reconfiguration that allows independent reconfiguration of individual pipeline stages. To enable efficient implementation of mixed-precision applications, the capability to dynamically fracture the internal compute-path of the DSP block is also incorporated into the design.

The design automation tool for the NATURE platform is extended to enable efficient mapping of compute intensive kernels utilizing the proposed DSP architecture(s) by exploring optimum resource sharing and area/power reduction. A design space exploration algorithm is developed and incorporated into the mapping tool that can determine the optimal configuration for a given input circuit, based on the design requirements and user constraints. The proposed technique automatically explores the different folding levels and DSP modes (configurations), evaluates their area/power trade-off and determines the most efficient mapping of the chosen configuration, which is subsequently fed to the mapping flow to generate the bitstream.

The contributions of this work would allow system designers to design and map compute intensive arithmetic kernels on the next generation hybrid multi-context FPGA platforms with ease, while providing high computational performance and energy efficiency that are required for many modern applications.

Acknowledgments

It is my pleasure to acknowledge the roles of several individuals who were instrumental in the completion of my PhD research.

First and foremost, I would like to thank Dr. Sudha Natarajan for giving me an opportunity to pursue PhD at Nanyang Technological university, Singapore. I would like to express my deepest gratitude to my supervisors Prof. Vun Chan Hua, Nicholas and Prof. Wei Zhang who have given me invaluable guidance, encouragement and support in many aspects of my research. Without Prof. Vun constant support and timely advice, I could not have completed my thesis on time. Prof. Zhang's charming attitude and enthusiasm have motivated me to a large extent to carry out my research with confidence.

I would like to thank Prof. Vinod A Prasad, my mentor and idol for the day-to-day insightful conversations and guidance. Selfless sharing of his own life experiences and his vast knowledge has influenced my research and life to a great extent. I am indebted to my best friend and my inspiration, Parvathi Balachandran, for her invincible support without which I would not have reached this level. Her generous support and motivation always helped me to overcome all dubious situations that I faced during my research period.

I take this opportunity to thank my colleagues and friends at Hardware & Embedded Systems Lab (HESL), especially Ronak Bajaj, Abhishek Jain, Supriya Satyanarayana, Prashob Nair, Abhishek Ambede, Sumedh Dhabu, Narendar Madhavan, Vipin Kizheppatt, Jiang Lianlian, Dr. Kavitha Jubin and Dr. Smitha Sreekumar for their invaluable support and encouragement. My gratitude to our lab executive, Chua Ngee Tat, who was always ready to help whenever I faced software related issues. A special thanks to Shreejith for his inspiration and whole hearted support provided during my work. I express my deepest gratitude to Neethu Robinson, for the precious love, care, companionship and for all the reasons to smile. I am also grateful to my roommates, friends and their family, Amrith, James, Rahul Warrior, Jithin, Tijo, Jithin Mathew, Jubin, Rashob, Renjith, Ashish, Lijesh and Anoop who all filled my stay in Singapore with happiness, good humour, and sustained positivity that helped me get through the highs and lows of my PhD journey.

Finally, I would like to thank my family: my parents, brother, grandmother, and in-laws for supporting me with their constant love and prayers throughout my studies. Their support and trust in my abilities helped me to aim high and cross

all tedious barriers to reach my goal. Last, but not least, I would like to express my deepest gratitude to my wife, for her patience, love and care. The thesis could not have been completed without her constant support and encouragement.

Contents

List of Figures

List of Tables

List of Abbreviations

A-D	Area Delay product
ALU	Arithmetic Logic Unit
ARF	Auto Regression Filter
ASPP	Application Specific Programmable Processor
BRAM	Block Random Access Memory
BW	Baugh-Wooley
CB	Connection Block
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide Semiconductor
DCT	Discrete Cosine Transform
DFF	D Flip Flop
DFG	Data Flow Graph
DPGA	Dynamically Programmable Gate Array
DSP	Digital Signal Processing
EWF	Elliptical Wave Filter
FDR	Fine-grained Dynamically Reconfigurable architecture
FDS	Force-Directed Scheduling
FF	Flip-Flop
FFT	Fast Fourier Transform

FIR	Finite Impulse Response
FL	Folding Level
FPGA	Field Programmable Gate Array
GCD	Greatest Common Divisor
GPU	Graphic Processing Unit
HDL	Hardware Description Language
HPC	High performance Computing
HPM	High performance Multiplier
IIR	Infinite Impulse Response
LB	Logic Block
LE	Logic Element
LUT	Look-Up Table
MAC	Multiply-and-Accumulate
MB	Macro Block
MB	Modified Booth
MUX	Multiplexer
NATURE	CMOS/NAno Technology reconfigURablE architecture
NRAM	Nano Random Access Memory
P-D	Power Delay product
PP	Partial Product
PR	Partial Reconfiguration
RTL	Register Transfer Level
RTR	RunTime Reconfiguration
SB	Switch Block
SMB	Super Macro Block

TSMC Taiwan Semiconductor Manufacturing Company

VPR Versatile Place and Route

1

Introduction

Over the past decade, the computational complexity of applications have grown multifold, making it increasingly difficult to extract the required performance using conventional processors. One of the most promising solutions is the adoption of hardware accelerators like Field Programmable Gate Arrays (FPGAs). This is because FPGAs are able to provide the performance close to Application Specific Integrated Circuits (ASICs) while maintaining the design flexibility of General Purpose Processors (GPPs), along with shorter time-to-market and reduced power consumption. These features make FPGA a better proponent for realizing compute intensive kernels.

Conceptually, all FPGA devices can be considered to be composed of two distinct layers: the configuration memory plane and the hardware logic plane. FPGAs achieve their unique re-programmability as well as flexibility by this composition.

The hardware logic plane consists of a set of reconfigurable blocks like Look-Up-Tables (LUTs), Flip-Flops (FFs), Digital Signal Processing (DSP) blocks, Block Random Access Memories (BRAMs) etc., that are connected via programmable interconnects. By configuring the logic blocks with appropriate interconnects, any digital circuit that fits within the available resources can be implemented on an FPGA platform. The configuration memory plane stores the FPGA configuration bitstreams. In conventional FPGA fabric, the configuration memory plane is usually Static RAM (SRAM) based and hence volatile.

Early generations of FPGA architectures supported only single-context and static reconfiguration, thereby permitting only one full-chip configuration to be loaded at a time. This style of reconfiguration required the FPGA to be powered down for every configuration change, making them unsuitable for high performance applications that demand fast runtime switching of functionality. One way to address this issue is to swap the configuration bits stored in the memory in and out of the hardware as they are needed during application execution. This is known as runtime reconfiguration (RTR) or dynamic reconfiguration [11], where the functionality is changed by altering the contents of configuration memory or by swapping the configuration plane. This RTR technique increases the logic density of FPGA as well as reduces the reconfiguration time. Various dynamic reconfiguration models such as single-context, multi-context, and Partial Reconfiguration (PR) are proposed in the literature.

Single-context reconfiguration uses only a single configuration memory plane that can hold one bitstream [12, 13], which can be programmed using a serial stream of configuration bits. Since it supports only sequential access, even a minor change in the application demands a complete reprogramming of the entire chip and the reconfiguration delay is in the range of milliseconds. Another type of RTR based FPGA platform is the multi-context reconfigurable architecture. In this approach, different applications can be assigned to different contexts. The configuration bits corresponding to each context are stored in different configuration memory planes [14, 15, 16]. A multi-context FPGA platform can be viewed as a multiplexed set of single-context devices. This system allows an inactive configuration plane

to be loaded with the next circuit configuration, while the previous circuit is in operation.

PR is a comparatively new reconfiguration technique supported by FPGAs, especially those from Xilinx Inc. It takes advantage of the fact that the FPGA bitstream is stored in a configuration memory with individually addressable content locations. By selectively modifying the contents of the memory, portions of the design can be altered while the remaining parts of the circuit continue to operate without interruption [17, 18]. However, in all these dynamic reconfigurable architectures, SRAM cells are used to store the configuration bitstream. Two main challenges that limit the wide adoption of such SRAM based dynamic reconfigurable architectures as an efficient hardware accelerators, especially for applications which demand frequent reconfiguration, are its long reconfiguration latency and limited on-chip storage.

With technology scaling towards nanoscale, researchers have been working towards development of nanodevices such as carbon nanotubes, nanowires, etc., that inherit the characteristics of nanotechnology in terms of integration density, performance, and power consumption. Based on these nanodevices, novel reconfigurable nanoarchitectures have been proposed. Various hybrid reconfigurable architectures that use nanowire-based Field Effect Transistors (FETs) and reconfigurable switches for improving the performance and logic density are described in the literature [19, 20, 21]. Hybrid CMOS/nano reconfigurable architecture exploit the benefits of both CMOS technology and nano technology. The use of nano RAMs like nanotube RAM (NRAM) [22], Phase-Change Memory (PCM) [23], Magnetoresistive RAMS (MRAM) [24] etc., as configuration planes in an FPGA platforms enable extremely fast reconfiguration (of the order of pico seconds) and increased logic density compared to a SRAM based FPGA architecture. One such multi-context hybrid architecture is the CMOS/NAnoTechnology reconfigURablE architecture, called NATURE architecture [8], which integrates nano RAMs as on-chip storage elements efficiently with CMOS logic. High density nonvolatile NRAMs are distributed throughout the NATURE architecture to allow large on-chip configuration storage enabling fine-grained cycle-by-cycle reconfiguration. During runtime

reconfiguration, the bitstreams are dynamically loaded from the memory layer (nano RAMs) to the CMOS logic layer which consists of fine-grained elements like LUTs and FFs to support the current computation. NATURE supports temporal logic folding which significantly improves logic density and area-delay product and provides design flexibility in performing area-delay trade-offs. Hence, compared to a traditional SRAM based multi-context FPGA platform, hybrid multi-context dynamically reconfigurable architecture provides significant improvements in area-time product making it a better choice for compute intensive kernels that can take advantage of its fast context switching.

1.1 Motivations

One of the major bottlenecks that limit the extensive usage of a hybrid multi-context architecture like the NATURE for efficient high performance computing is its fine granularity. During technology mapping, all compute intensive arithmetic and other operation nodes are realized using LUTs and FFs. Most of the arithmetic circuits used in DSP applications consist of a large number of multipliers and multiply-and-accumulate (MAC) computations. Digital communication circuits and image processing algorithms such as FIR filter, compression/decompression algorithms are a few examples of such compute intensive applications. Realizing such complex applications using the NATURE architecture results in reduced computational performance due to the fine granularity of the architecture. This is justified by the fact that realizing a 16-bit multiplier node using the NATURE platform would consume 488 LUTs and incur a latency of 18.6 ns for each computation. This also results in large area utilization due to the usage of a large number of LUTs and FFs to realize such circuits.

On the other hand, researchers have shown the benefits of coupling coarse-grained blocks (DSP) with fine-grained logic blocks, which can also be observed in commercial FPGAs from Xilinx and Altera [5, 2]. These DSP blocks have been designed

specifically targeting their respective platforms that support runtime reconfiguration. Although, some of these DSP blocks support dynamic reconfiguration, efficient reuse of these DSP blocks requires deep knowledge about their architectures and have to be explicitly handled in the user design.

For improving the compute capability on multi-context FPGAs like NATURE, efficient reconfigurable DSP architectures need to be designed for mapping complex arithmetic operations. Also, these DSP blocks should support cycle level dynamic reconfigurability to fully exploit the capability of the NATURE architecture. DSP blocks that support cycle level reconfiguration allows efficient reuse of hard blocks thereby achieving reduced resource utilization and power consumption.

Efficient and optimal mapping of input circuit designs on FPGA require an equally efficient mapping tool. Typically, the design optimization tool flow verifies the functionality and timing constraints, performs mapping and scheduling of circuit nodes and finally generates the configuration bits after placement and routing. It is required to extend the tool flow according to the FPGA architectural enhancement for efficient mapping. Enhancing the architecture of NATURE cannot be done exclusively without augmenting its dedicated design automation tool flow called, NanoMap [10]. The existing NanoMap tool performs logic mapping, temporal clustering, temporal placement and routing and, finally generates a configuration bitmap for the fine-grained NATURE architecture. However, it is required to extend the NanoMap tool flow for efficient mapping of compute intensive arithmetic nodes on the coarse-grained blocks in the NATURE platform. Further, the temporal clustering algorithm needs to be enhanced for clever packing of arithmetic operations onto the DSP blocks, resulting in reduced coarse-grained block utilization and power consumption. Corresponding to the architectural enhancement of NATURE, a careful investigation of the placement and routing tools has to be performed in order to determine the best allocation for DSP blocks for flexible routing.

Growing demand for portable communication devices and mobile computing platforms demand the need for energy efficient mapping of compute intensive kernels

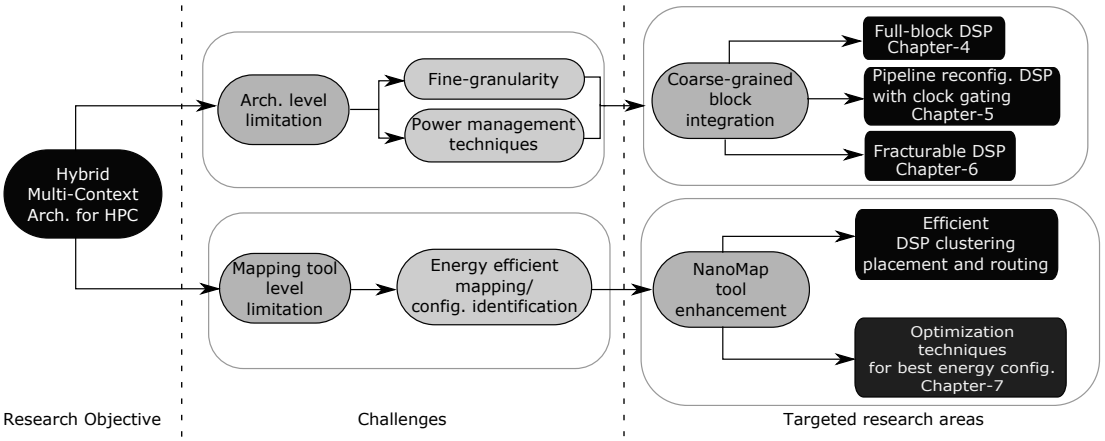


Figure 1.1: Outline of challenges in NATURE architecture addressed in this thesis and the research objectives.

on hardware accelerators. Many researchers have been putting effort at the architecture and Computer-Aided Design (CAD) tools for FPGAs to make them an energy efficient platform for high performance computing [25, 26, 27]. Moreover, with the kind of flexibility available in reconfiguration parameters provided by mixed-grained multi-context FPGAs, it is important to choose the best configuration parameter for energy efficient mapping of a design based on constraints imposed by the designer.

1.2 Objectives

The major research goals of this thesis are indicated in this section. The primary objective of the thesis is to enable the next generation of multi-context FPGAs like NATURE to be used as an efficient platform for high performance computing. To meet this goal, the research work explores and presents how DSP architectures can be designed for the NATURE platform in order to fully exploit its advantages and possibilities to map complex arithmetic operations. For efficient mapping of an application to the mixed-grained NATURE architecture, an intelligent design automation tool is required. As such, the NATURE's NanoMap tool flow needs to be extended to enable efficient mapping of compute intensive kernels utilizing the proposed DSP architecture(s) by exploring resource sharing and area/power reduction. Further, to enhance the NATURE architecture for implementing energy

efficient reconfigurable systems, a design space exploration algorithm needs to be incorporated into the mapping tool that can determine the optimal configuration (folding level and DSP modes) for a given input circuit, based on the design requirements and user constraints. The major research objectives are indicated in Figure. 1.1 and are listed below.

- To enable new generation hybrid multi-context FPGA suitable for HPC:
  - Architectural level exploration:
    1. To explore DSP block architectures for the hybrid multi-context reconfigurable platforms in order to fully exploit its advantages and capabilities.
    2. Incorporate efficient power management techniques for optimal energy utilization.
  - Mapping tool level exploration:
    1. To extend the NanoMap design automation tool flow by adding the capability to efficiently map complex arithmetic operations on DSP incorporated NATURE architecture.
    2. To develop intelligent algorithms that can determine the optimized configurations for the energy efficient configuration for a design based on constraints imposed by the designer.

The above mentioned research challenges will be addressed in this thesis through the design and development of efficient DSP incorporated in the NATURE architecture, targeted for compute intensive applications like digital communication circuits and image processing algorithms.

1.3 Contributions

The main contributions of this thesis are the reconfigurable coarse-grained DSP architecture, its design features and reconfigurability that can be used in next generation hybrid multi-context FPGAs like the NATURE platform. The thesis also contributes to the development of an automated design tool flow for optimal energy efficient mapping of circuits for a given user constraint on the NATURE architecture. The design automation tool flow, NanoMap is extended for efficient mapping of compute intensive applications on DSP incorporated in the NATURE architecture. The specific contributions are listed below:

1. A comprehensive study is performed on the coarse-grained blocks from the perspective of dynamically reconfigurable architecture, specifically with the NATURE architecture for high performance computing. Three novel DSP architectures optimized for the NATURE platform are proposed as the result of this investigation.

   (a) Full-block DSP: This is a dynamically reconfigurable DSP block, which is fully reconfigurable during runtime to implement compute intensive arithmetic operations in different clock cycles. A route through DSP technique is also introduced that offers flexibility for the designers to exploit the features of the DSP block.

   (b) Pipeline Reconfigurable DSP: This is an enhanced version of the proposed full-block DSP architecture. It allows the reconfiguration of individual pipeline stage. A multi-stage clock gating techniques are also used to incorporate efficient power consumption within the coarse-grained blocks.

(c) Fracturable DSP block: This is a variable precision DSP block architecture capable of dynamically fracturing the internal compute-path to enable efficient implementation of mixed precision applications.

2. An efficient mapping algorithm on mixed-grained NATURE architecture is developed in this work. This work is integrated into NATURE's NanoMap tool to demonstrate its effectiveness for the proposed DSP blocks. Efficient algorithms for optimal packing of DSP blocks followed by enhancement of Versatile Place and Route (VPR) tool flow for the best placement and routing are also proposed in this thesis.

3. A design space exploration algorithm that can determine the best folding level and depth relaxation for better power-delay/area-delay trade-off to map a design based on designer constraints.

1.4 Thesis Organization

The remaining thesis is organized as follows: Chapter 2 gives an overview of the related works in reconfigurable architectures. It also presents an overview of recent research works on commercial and academic DSP architecture designs, their advantages and comparisons. Chapter 3 presents the background study of a fine-grained multi-context FPGA architecture, the NATURE platform and its mapping tool (NanoMap) which is used in this research work to demonstrate the performances and effectiveness of the various techniques proposed in this thesis. Chapter 4 presents the proposed coarse-grained Digital Signal Processing (DSP) block, its detailed architecture, interconnect design, reconfiguration of DSP, applications and experimental results. Chapter 5 describes the pipeline reconfiguration DSP architecture that exploits NATURE's temporal logic folding capability to further augment its performance. Chapter 6 presents the technique that can be used to execute multiple sub-width arithmetic operations simultaneously on a single DSP block by dynamically switching between sub-width and full-width operations. Chapter 7 presents an automated design tool flow for the NATURE

platform which maps the input circuit using the best energy efficient configuration for a given user constraint. Chapter 8 then concludes the work presented in this thesis and outlines the future research directions.

1.5 Publications

Most of the work presented in this thesis has been written up in a number of published and submitted papers listed below:

Journal articles

1. R. Warrier, S. Shreejith, W. Zhang, C. H. Vun, S. A. Fahmy, *Fracturable DSP Block for Multi-Context Reconfigurable Architectures*, Springer Journal on Circuits, Systems and Signal Processing (CSSP), November 2016, pp. 1-14.

2. R. Warrier, W. Zhang, C. H. Vun, *Pipeline Reconfigurable DSP for Dynamically Reconfigurable Architectures*, Springer Journal on Circuits, Systems and Signal Processing (CSSP), (Accepted).

3. R. Warrier, W. Zhang, C. H. Vun, *Power-Aware High-Level Synthesis for Dynamically Reconfigurable Architectures*, IEEE Transaction on Computer-Aided Design (TCAD) (prepared for submission).

Conference proceedings

4. L. Hao, S. Sinha, R. Warrier, W. Zhang, *Static Hardware Task Placement on Multi-Context FPGA Using Hybrid Genetic Algorithm*, in International Conference on Field Programmable Logic and Applications (FPL), September 2015, pp. 1-8.

5. R. Warrier, C. H. Vun, W. Zhang, *A Low-Power Pipelined MAC Architecture Using Baugh-Wooley Based Multiplier*, in Proceedings of the IEEE Global Conference on Consumer Electronics (GCCE), October 2014, pp. 505-506.

6. R. Warrier, L. Hao, W. Zhang, *Reconfigurable DSP Block Design for Dynamically Reconfigurable Architecture*, in Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), June 2014, pp. 2551-2554.

2

Literature Survey

In this chapter, an overview of the generic FPGA architecture, routing interconnects and conventional mapping tool flow is presented. Prior works on commercial and academic FPGA designs are discussed in detail. Special focus is given to architectural features and capabilities of both commercial and special purpose DSP blocks that are available in the literature. Various design methodologies and optimization schemes employed in different FPGA architectures for efficient mapping of circuit designs are also investigated. With focus shifting towards low power FPGA implementations, the chapter concludes with a discussion on various methodologies employed at both architecture and tool level for energy efficient mapping of input designs on reconfigurable platforms.

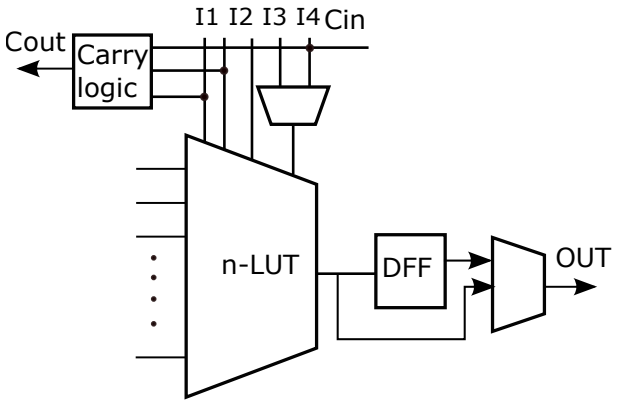


Figure 2.1: Basic fine-grained logic block.

2.1 Reconfigurable Architectures

A reconfigurable architecture consists of a set of configurable compute blocks, programmable interconnects and a flexible interface (input/output pads) to interact with the rest of the system. In this section, we review the different FPGA architectures and summarize their features.

2.1.1 Conventional FPGA Architecture

The basic building block in an FPGA is a set of Configurable Logic Blocks (CLBs), arranged in an island style configuration. In a fine-grained architecture, the logic block contains only very basic logic, implementing a single function on a small number of bits. Figure 2.1 shows a typical fine-grained block, consisting of an n-input LUT, carry chain, a D-Flip Flop (DFF) and multiplexers (MUXs). A LUT is a small memory-like element usually 1-bit wide and 16 or 64 bits deep. By storing appropriate values in these elements, any Boolean function can be implemented. DFFs allow temporary storage of values and implementation of synchronous computation. The fast carry logic is a special resource to speed up carry based computations [11].

On the other hand, a coarse-grained blocks consist of Arithmetic and Logic Units (ALUs), DSPs and even BRAMs. BRAMs and DSPs are arranged in column fashion spread across the FPGA. The DSP blocks can also be connected with

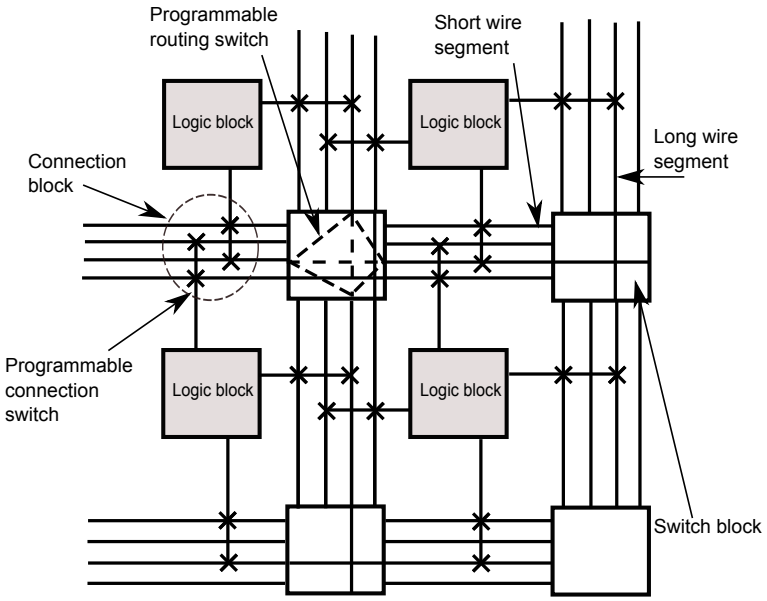


Figure 2.2: Routing resource and connectivity reproduced from [1].

dedicated interconnect lines which can be efficiently used when cascading multiple DSP blocks. BRAM blocks are generally incorporated within the architecture so that temporary results can be stored. Input/ Output (I/O) blocks are connected at the periphery of the grid, which can be connected to CLBs and DSPs depending on the required logic. In addition to the coarse-grained and fine-grained elements, there are SRAMs distributed in the reconfigurable plane to store the bitstreams (configuration bits).

With the advancement in technology, current generations of FPGAs are often also integrated with hard processors, such that general purpose logic can be easily replaced by them, together with glue logic implementation in FPGAs. Hence, although a System on-Chip (SoC) typically uses an ASIC implementation, today's FPGAs can also be used to provide a SoC solution.

FPGA Interconnect

Figure 2.2 shows the routing architecture used by a typical FPGA. The layout is mainly island style with general purpose routing channels surrounding both fine-grained and coarse-grained blocks. Routing resources incorporate wire segments of varying lengths, where long wires span multiple logic blocks while short wires span

into a few logic blocks depending upon the wire segment length. The Connection Blocks (CB) [28] enable connectivity between I/O pins of CLBs/DSPs/BRAMs and the selected wire segments or all the adjacent wire segments. Switch Blocks (SB) [28] are provided at the intersection of every horizontal and vertical channel. These are a set of programmable switches that allow some of the wire segments incident to the SB to be connected to others. By configuring appropriate switches, short wire segments can be connected together to form long connections. The signal flows from the logic block into the CB then continue into the routing channel, and eventually to the SB which provides for a change in direction. The CB and SB contain reconfigurable MUXs or pass transistor switches.

Conventional FPGA Mapping

The process of implementing a design on an FPGA starts with describing the design in a HDL like Verilog or VHSIC Hardware Description Language (VHDL) and ends with a stream of bits, which is loaded into the FPGAs configuration memory. The configuration memory controls all the low level features of the fabric, determining the logic contents of the LUTs, how all primitives are connected, and which features are used. With the increase in complexity of circuit designs, researchers have also explored the possibilities to efficiently describe such systems at a higher level of abstraction which will improve the design and verification tasks. This includes research into tools that can convert high-level languages like SystemC, C and C++ into hardware. Generally, these tool takes the design description in a high-level language and translate them into synthesizable code which can be used in further steps (logic mapping, scheduling, placement and routing) of the generic tool flow. After a circuit is described in HDL, the initial step of the tool flow is to verify the functionality of the design. Typically, there are three major steps involved in the generation of bit streams from a Verilog design level. These steps are:

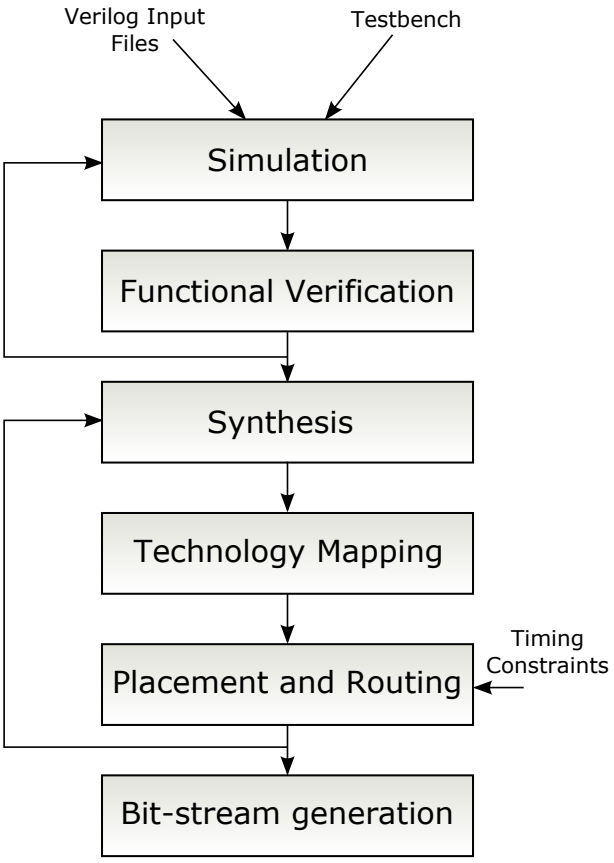


Figure 2.3: Typical FPGA design flow.

- Synthesis
- Technology mapping
- Placement and Routing

The flow diagram of a typical FPGA implementation process is shown in Figure 2.3.

- *Synthesis:* In this process the input file, which is in HDL format is transformed into a network of basic building blocks, including LUTs for Boolean logic, FFs for synchronous components, other basic arithmetic units, and in modern tools, even to some device specific hard blocks. Different optimization techniques are employed to minimize the number of logic gates thereby reducing the total area and circuit delay of the final implementation of the design. The output of the synthesis stage is a network of Boolean gates, FFs,

basic circuits, hard blocks (DSPs, BRAM) and wiring connections between all these basic components in the network.

- *Technology mapping:* It is defined as mapping of the network to a given set of pre-defined library cells. These library cells comprise LUTs, FFs, basic arithmetic circuits like adders, multipliers and other advanced hard blocks. Hence the technology mapping consists of segmenting the Boolean network into a set of nodes, where each set will be mapped to one of the basic blocks in the library cells.

- *Placement and Routing:* The placement algorithms determine which logic block within an FPGA should be used to implement which part of the logic required in the circuit. There are optimization methods employed in this process such as to place the connected logic blocks as close as possible to minimize the required wiring. In another placement method where routability is given priority, the wiring density would be balanced across the FPGA. In the time-driven placement method, the preference is given to minimization of delay of the entire circuit. The three main classes of placers used today are min-cut (partitioning-based) [29, 30, 31], analytic [32, 33, 34, 35, 36] and simulated annealing [37, 38, 39, 40].

  After the placement process, in which the physical placement of logic blocks in a circuit have been decided, a router decides which routing switches have to be turned on in order to connect between each logic block input and output pins as required by circuit. Generally, a directed graph [41, 42] are used to represent the routing architecture of FPGA. In this graph, each wire and each logic block pin becomes a node and potential connections become edges. FPGA routers can be classified into two groups: Combined global-detailed routers [43, 44, 45, 46, 47] that take only one step to determine a complete routing path, and the two step routing algorithm, which first performs global routing followed by detailed routing. Global routing determines the appropriate nets to be used with the various logic block pin and channel segments while detailed routing [48, 49, 50, 51] determines the wire(s) each net will use within each channel segment.

After successful execution of all these steps, a bitstream will be generated that is to be stored in the configuration memory of FPGA.

2.1.2 Commercial FPGA

Profiling the top FPGA vendors of recent years, Xilinx and Altera continue to dominate the market for general purpose programmable logic [52, 53] with approximately 90% of the market share (Xilinx 49% and Altera 41%). However, a few companies like Atmel, Lattice semiconductor, Actel, Tabula etc are also gradually winning market share by targeting niche applications. Atmel uses Electrically Programmable Logic Device (EPLD) technology to develop their product lines [54], while Actel devices [55] use anti-fuse technology which is qualified for both military and spaceflight applications. Tabula uses slightly different terminology to describe their devices (ABAX device family), referring to Spacetime technology to achieve high logic density and area saving architectures [56].

(i) Xilinx

Xilinx has the largest programmable logic portfolio in the industry. They have a good range of FPGAs in terms of cost and performance. Built on 45 nm technology, Spartan-6 devices are the most cost-optimized FPGAs that are ideally suited for a range of advanced bridging applications found in infotainment, consumer and industrial automation. Their current FPGA family consists of the Series-7 (Artix-7, Kintex-7, Virtex-7, Spartan-7) which are built on a 28 nm process. Artix and Kintex series which provide better coverage of lower cost and power targeted for mid-range applications. For highest system performance Virtex-7 family provides advanced high performance FPGA logic. They are the basic logic blocks for implementing sequential as well as combinational circuits in Xilinx FPGA architecture. The CLB used in 7-series is identical to that in the Virtex-6 FPGA family. Each

Table 2.1: Xilinx multi-node product portfolio.

45nm	**28nm**	**20nm**	**16nm**
Spartan-6	Virtex-7 Kintex-7 Artix-7 Spartan-7	$\text{Virtex}_{UltraScale}$ $\text{Kintex}_{UltraScale}$	$\text{Virtex}_{UltraScale+}$ $\text{Kintex}_{UltraScale+}$

CLB element is connected to a switch matrix for accessing a general routing matrix. A CLB contains a pair of slices consisting of four 6-input LUTs and eight storage elements.

In addition to SRAM based FPGA products, they also offer Flash and EE type CPLD devices (CoolRunner-II and XC9500XL). Xilinx Zynq-7000 SoC is a new hybrid reconfigurable device that support partial reconfiguration couples a powerful ARM Cortex A9 processor and 28 nm Xilinx programmable logic that provides a better platform for designing smarter systems with tightly coupled software based control and logic with real time hardware based processing and optimized system interfaces. The ARM processor in Zynq devices communicates with on-chip memory, memory controllers and peripheral blocks through AXI interconnect. Xilinx extended their device range to UltraScale portfolio spanning 20 nm and 16 nm FPGA, SoC and 3D IC devices and leverage a significant boost in performance. To enable highest level of performance and integration, the UltraScale$^+$ family (16 nm) also includes a new interconnect optimization technology, SmartConnect.

Table 2.1 shows the Xilinx multi-node product portfolio.

(ii) Altera

Altera's products include FPGA, CPLD, and SoC. "Stratix" series is their high end FPGA family. In the mid-range is their "Arria" series and the "Cyclone" series rounds out their low cost offering (Arria-V, Cyclone-V and Stratix-V build on 28 nm process technology). Variation of Arria and Cyclone are available as SoC integrated with hard silicon core processors.

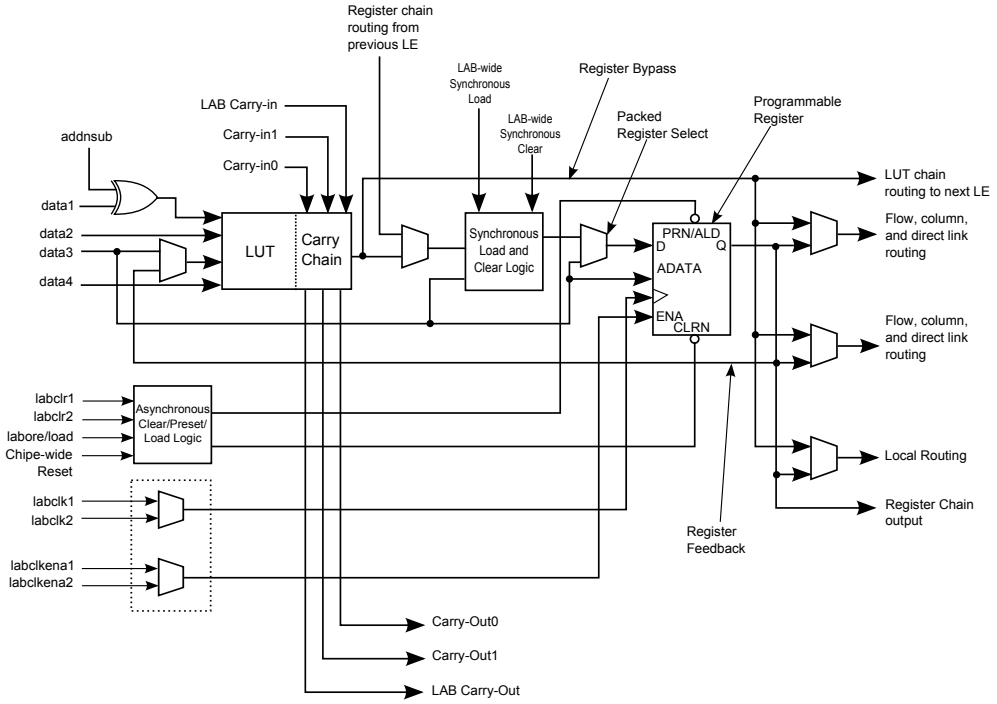


Figure 2.4: Stratix LE reproduced from [2].

The Stratix device employs a 2-D architecture to implement custom logic [2]. Logic Array Block (LAB), memory and DSPs are interconnected using routing resources of varying length and speed. The logic array consists of LABs with 10 LEs in each LAB. Each LE unit provides efficient implementation of user logic functions, containing a 4-input LUT, a programmable register and a carry chain with carry select capability. It also supports dynamic single bit addition or subtraction selectable by a LAB-wide control signal. LABs drive all types of interconnect: local, column, row, LUT chain, register chain and direct links. The Stratix LE structure is shown in Figure 2.4.

A summary comparing Altera's FPGA families [13] is shown in Table 2.2.

2.1.3 Coarse-Grained Reconfigurable Architecture

These are reconfigurable architectures that are designed for specific types of applications. Unlike commercial FPGAs that consist of both fine-grained and coarse-grained blocks, these specially-designed FPGAs usually contain only coarse-grained

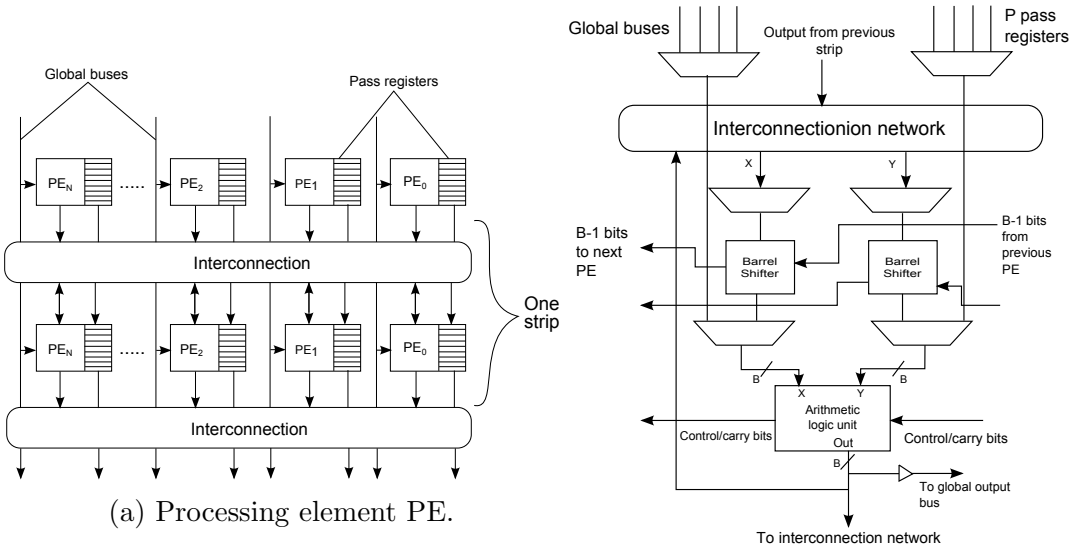
Table 2.2: Comparison of Altera FPGA families.

Features	**Arria-10**	**Arria-V**	**Stratix-V**	**Cyclone-V**
Adaptive logic module ALM	427,700	190,240	359,200	113,560
LE	1150,000	504,000	952,000	301,000
Registers	1,710,800	760,960	1,436,800	454,240
M20K memory blocks	2,713	2,414	2,640	1,220
M20K memory (Mb)	54	24.1	52	6.1
MLAB memory (Mb)	12.7	2.9	10.96	0.8
DSP	1,518	1156	352	342

blocks. The main advantage of such architectures are a reduction in configuration memory and configuration time.

In a mesh-based architecture, the Processing Elements (PEs) are arranged as rectangular 2-D arrays with horizontal and vertical connections which enable efficient communication resources for significant parallelism. The Colt architecture [57], derived from an amalgamation of FPGA and data flow computing concepts [58] is a 16 bit pipenet [59] and relies on runtime reconfiguration using wormhole routing. The Garp architecture which mimics an FPGA fabric consists of 32 PEs arranged in rows to form a reconfigurable ALU [60]. It has a MIPS-II like host and for acceleration of specific loops, a 32×24 RA of LUT based 2 bit PEs is used. Another special architecture is MorphoSys [61] which has a TINYRISC processor with an extended instruction set, an 8×8 reconfigurable arrays, a DMA controller, context memory and a frame buffer for intermediate data. The CHESS array [62] resembles a chessboard like floor plan with interleaved rows of alternating ALU/ switch box sequence. The high memory requirement is supported by embedded RAM.

The Reconfigurable Pipeline Datapath (RaPiD) and PipeRench architectures fall under the linear array based architectures. Using deep pipelines, RaPiD [63] aims to speedup highly compute intensive tasks. It offers different computing resources like ALUs, RAMs registers, and multipliers, but which are irregularly distributed. For accelerating pipelined applications, PipeRench [3] provides several reconfigurable pipeline stages (stripes) and support fast partial dynamic reconfiguration



(a) Processing element PE.

(b) Detailed View.

Figure 2.5: Piperench architecture reproduced from [3].

and runtime scheduling of configuration streams and data streams. The reconfigurable fabric allows the configuration of a pipeline stage in every cycle subsequently allowing the parallel execution of all other stages. The fabric contains a set of physical pipeline stages, called stripes as shown in the Figure 2.5. Each stripes has an interconnection network and a set of PEs. Each PE contains an ALU and a pass register file. Each ALU contains LUTs and extra circuitry for carry chains, zero detection, and so on. Wider ALU operations are performed by cascading carry lines of PipeRench and complex combinational functions are realized by chaining PEs together via interconnection network. While RaPiD generally supports static reconfiguration, PipeRench relies on dynamic reconfiguration allowing the PEs to be reconfigured in each execution cycle.

There are various reconfigurable arrays that are based on crossbar switch based designs. Programmable Arithmetic Device for DSP (PADDI-1) [66] is used for rapid prototyping of computation-intensive DSP data paths. It features eight PEs all connected by a multilayer crossbar. The PADDI-2 [67] architecture has 48 PEs, with each featuring a 16 bit ALU. It saves area by using a restricted crossbar with a hierarchical interconnect structure for linear arrays of PE forming clusters. The

Table 2.3: Summary of coarse-grained reconfigurable architectures.

Style	Device	Architecture	Source	Granularity	Fabrics	Mapping	Target application
mesh	DP-FPGA	2-D array	[64]	1 & 4 bit multi-granular	inhomogeneous routing channels	switchbox routing	regular datapaths
	Colt	2-D array	[57]	1 & 16 bit inhomogeneous	sophisticated	RTR	highly dynamic reconfiguration
	Garp	2-D mesh	[60]	2 bit	global & semi-global lines	heuristic routing	loop acceleration
	MorphoSys	2-D mesh	[61]	16 bit	NN, length 2 & 3 global lines	manual P&R	not disclosed
	REMARC	2-D mesh	[65]	16 bit	NN,full length &	manual P&R	multimedia
linear	RaPiD	1-D array	[63]	16 bit	segmented buses	channel routing	pipelining
	PipeRench	1-D array	[3]	128 bit	sophisticated	scheduling	pipelining
crossbar	PADDI	crossbar	[66]	16 bit	central crossbar	routing	DSP
	PADDI-2	crossbar	[67]	16 bit	multiple crossbar	routing	DSP and others
	Pleiades	mesh/crossbar	[68]	multi-granular	multiple segmented crossbar	switchbox routing	multimedia

Pleiades architecture [68] is a low power PADDI-3 with programmable microprocessor and heterogeneous coarse-grained PEs in which communication is data-flow driven.

REMARC architecture consists of a MIPS ISA based core and an 8×8 reconfigurable logic array [65]. Each processing element of the array consists of a 16-bit processor and the instruction stored in small local memory control the execution of each processor. Virtual Embedded blocks (VEBs) was proposed to explore hard logic integration in an FPGA array [69]. Researchers proposed a methodology to study the effect of embedding floating point coarse-grained units in FPGAs. Experimental results show that embedding coarse-grained floating point unit in FPGAs can result in 3.7× area reduction and 4.4× speedup. Chameleon [70] is another architecture that consists of Montinum Tile Processor, which has a cluster of 5 ALUs, each fed by two register files and all five controlled by a single and simple sequencer. Zippy, a coarse-grained multi-context hybrid CPU with architectural support for efficient hardware virtualization [71]. The Zippy model integrates an embedded CPU core with a coarse-grained reconfigurable unit that can be parameterized to resemble whole families of hybrid CPUs. Besides the simulation tools, Zippy includes a tool-chain to generate software and hardware executables. SmartCell is a novel coarse-grained reconfigurable architecture which is targeted

for high data throughput and compute-intensive applications [72]. SmartCell provides stream processing capacity by integrating a large number of computational units with reconfigurable interconnection fabrics. SYSCORE is a low power novel CGRA architecture targeting signal processing in wearable and implanted devices. Configurable Function Units (CFUs) and Round About Interconnect (RAI) units are the two main elements of this architecture. Unlike the traditional processors, fetch-decode steps are eliminated in SYSCORE CGRA to provide significant energy savings. Systolic data reuse technique is used to reduce the number of intermediate data RAM accesses. An average of 62% energy savings is achieved by SYSCORE architecture over a conventional DSP and SIMD processor.

Table 2.3 shows a comparison of some coarse-grained reconfigurable architectures.

2.1.4 Dynamically Reconfigurable FPGAs

The development of dynamically reconfigurable architectures dates back to 1995, when R. T. Ong from Xilinx filed a patent for an FPGA which can store multiple configurations simultaneously [73]. The proposed design supports only two configuration memory arrays to store different configuration data. The time multiplexed FPGA, proposed by Trimberger [74], was an extension of the earlier version which support a maximum of eight configuration planes. In [75], Dehon introduced the concept of temporal pipelining to support dynamic reconfiguration in the Dynamically Programmable Gate Arrays (DPGAs). The work exploited the fact that the area of the instruction storage is much smaller than the area of a LUT. The logic density of the reconfigurable platform can be increased by reusing the LUT for different instructions at different times. However, the DPGA has limited logic capacity, operating frequency, and high power consumption. To overcome this limitation, Chong et al. [76] proposed a fine-grained reconfigurable architecture called Reconfigurable Context Memory (RCM) that exploits the regularity and redundancy in configuration bits between different contexts. Scalera and Vazquez [16] developed the first practical multi-context FPGA on a 0.35μm technology, called Context Switching Reconfigurable Computer (CSRC) that can store up to four

configurations concurrently. Garp was another dynamically reconfigurable architecture, that combined reconfigurable hardware with a standard MIPS processor [60]. Tabula also supported dynamic reconfiguration for their FPGA platform based on the concept of Spacetime technology, which enables rapid reconfiguration of hardware by fast context-switching of configuration bits that are locally stored [77]. Furthermore, Xilinx introduced the concept of dynamic partial reconfiguration, which allows the runtime reconfiguration of a small logic region. Although it significantly reduces the configuration time [78], the performance of Partial Reconfiguration (PR) is heavily impacted by design decisions (partitioning and floorplanning) [18, 79] and the long reconfiguration latency (in milliseconds). Altera also introduced PR on their Stratix-V series devices [80].

Various other dynamically reconfigurable FPGAs have also been proposed. The first challenge is the reconfiguration delay. Due to long reconfiguration delays, only 26 partial reconfiguration is possible at run-time. Hence, dynamic reconfigurations can only be used sparingly [74, 81, 82]. Other time-multiplexed architectures proposed in [83, 84, 85] also extend conventional FPGA architectures. Registers are added to store computational states and partial results. The LE in DRFPGA has four D flip-flops (DFFs) connected to form a shift register style temporal communication module (TCM), which stores intermediate results generated at different cycles. Fast run-time configuration is achieved by associating multiple SRAM cells to each configurable switch. However, only a few (4 to 8) contexts are allowed in these architectures because of the area overhead introduced due to the incorporation of a large number of SRAM cells per configurable switch. In addition, the rapid configuration rate can result in a power overhead [85, 86]. Miyamoto et al. addressed the power problem by using non-precharge low-power 9TSRAMs that support up to 16 contexts [87].

2.1.5 Hybrid Reconfigurable Architectures

The continuous advancement in process technology into the nanolevel gives researchers a new direction to develop nanodevices and nano circuits. Reconfigurable architectures based on nanodevices inherit their characteristics to achieve improved logic density, reduced power consumption etc. A reconfigurable architecture with programmable fabric made of nanowire connected using reconfigurable diode switches was proposed in [88]. A nanoscale fabric using nanowire-based FETs and reconfigurable switches was proposed in [20]. A hybrid cell-based reconfigurable architecture called CMOL, that consists of the CMOL stack as logic fabric which is connected using nanowires [89]. A hybrid CMOS/nano reconfigurable architecture with a crossbar of nanowires implementing logic and interconnects, and configuration interface implemented using CMOS logic was introduced by Rad et al. [90]. Other novel reconfigurable architectures based on nanodevices and nanocircuits have been proposed in the literature. These novel hybrid reconfigurable architectures achieve improvement in performance and logic density promising efficient alternatives when nanotechnology fabrication becomes practical.

FinFET has emerged as a promising replacement for the current CMOS technology due to its tighter control of the channel potential by the two gates wrapped around the body [91]. The front and the back gates in a FinFET can be used in various ways to enable different operation modes [92]. Controlling the two gates separately can create opportunities for innovative circuit designs with improved performance and power consumption [93, 94]. Beckett proposed a FinFET based low-power reconfigurable logic array containing homogeneous fine-grain reconfigurable cells [95, 96]. In each transistor, applying different voltages at the first gate configures the second gate to different operation points. Zhang et al. proposed another low-power architecture by using the two gates of FinFETs in a different way [97]. In the proposed design, the bias voltage applied to the back-gate is used to alter the threshold voltage (Vth) of the front gate, thereby controlling the leakage of the device.

2.2 DSP Architecture

Although signal processing is usually coupled with digital signal processors, it is becoming increasingly evident that FPGAs are taking over as the platform of choice in the implementation of high-performance, high precision signal processing. Accordingly, FPGA vendors are beginning to include hard multipliers and DSP blocks within their fabric. With the inclusion of hard DSP blocks in FPGAs, a wide range of signal processing applications (video surveillance, wireless base stations, medical imaging, military radar, etc.) that demands performance and precision can be realized. Typically, a DSP block can support large number of high precision arithmetic operations. DSP blocks are more power efficient, operate at higher frequency, and consume less area than the equivalent operations implemented using the fabric logic (LUTs).

2.2.1 Commercial DSP Architectures

(i) Altera Stratix DSP block for Multiple Applications

Altera's first generation of high-end FPGA families, Stratix FPGA device combined an architecture tuned for high performance. The first DSP block from Altera was introduced in Stratix FPGA, containing four 18×18 multipliers, accumulators, and a summation unit, with a maximum frequency of 333 MHz. The Stratix II and Stratix II GX device has two or four columns of DSP block that can be configured at different modes (eight 9×9, four 18×18, one 27×27, one 36×36), generating output at 450 MHz [98]. The DSP block in Stratix III device includes a dynamic shift function as an additional feature and operates at a maximum frequency of 550 MHz [98]. The logical functionality of the Stratix III DSP block is a superset of the previous generation of the DSP block found in Stratix and Stratix II devices. In Stratix IV device, the DSP block includes rich and flexible arithmetic rounding and saturation units and natively supported 18 bit complex multiplications, that operates at a frequency of 550 MHz. With variable precision DSP blocks, Altera

Table 2.4: Comparison of the DSP blocks in Altera FPGAs.

Devices	**Embedded Block**	**Speed**	**No of DSP Blocks**
Stratix II	18×18 DSP block	450 MHz	12 - 96
Stratix III	18×18 DSP block	550 MHz	27-96
Stratix IV	18×18 DSP block	550 MHz	384 - 1288
Stratix V	18×18 DSP block	500 MHz	512 - 3926
	Variable Precision DSP	550 MHz	256 - 1963
Stratix 10	Variable Precision DSP	550 MHz	1152 - 5760
	18×19 multiplier	-	2304 - 11520
Arria II	18×18 DSP block	550 MHz	232 - 1040
Arria V	Variable Precision DSP block	up to 550 MHz	240 - 1139
Arria 10	18×19	up to 550 MHz	312- 3316
Cyclone II	18×18 multiplier	-	13 - 150
Cyclone III	18×18 multiplier	-	23- 288
Cyclone IV	18×18 multiplier	-	40 - 266
Cyclone V	18×18 multiplier	up to 500 MHz	50 - 684
	Variable precision DSP		25 - 342
MAX 10	18×18 multiplier	-	16 - 144

Stratix V FPGA device supports various precisions ranging from 9×9 bits up to single-precision floating-point within a single DSP block [99]. The variable precision DSP block in Stratix V operates in 18-bit precision mode and high-precision mode, generating output at 550 MHz. The block delivers high performance due to the dedicated circuitry as shown in the Figure 2.6. High performance DSP block is included in Stratix 10 devices that can achieve up to 23 TMACs of fixed-point performance and up to 10 TFLOPS of IEEE 754 single-precision floating point performance [100]. With Stratix 10 devices, all fixed-point modes operate at a sustained 1 GHz frequency and all floating-point modes operate at a constant 800 MHz frequency.

Altera's Arria family incorporates DSP blocks with rich feature set delivering optimal performance and power efficiency in the midrange. The DSP block in Arria II FPGA device provides eight 18×18 multiplier, in addition to other native features

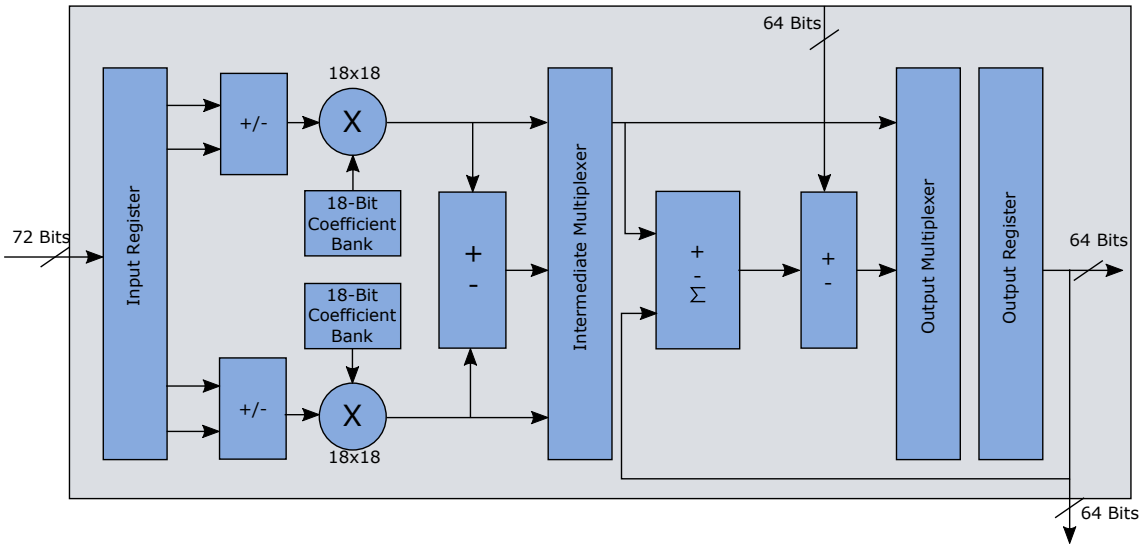


Figure 2.6: Variable precision DSP architecture reproduced from [4].

like registers, adders and subtractors etc, operating at a maximum frequency of 550 MHz. The DSP architecture of the 28 nm Arria V is optimized to support both high performance and variable precision data that enables area and power efficient implementation of both fixed and floating point operations. These DSP blocks can be configured at compile time into a 18 bit mode or in a high precision mode using dual 18×19 multiplier blocks. Supporting three operating modes (floating-point mode, standard-precision mode, high-precision mode), the variable precision DSP block included in Arria 10 FPGA delivers a maximum floating point performance up to 1.5 TFLOPS.

Cyclone devices provide an ideal platform for implementing low-cost DSP systems on an FPGA. 18×18 bit embedded multipliers are included in Cyclone II, III and IV devices. The number of embedded multipliers increases with device density. Similar to the Stratix V and Arria V devices, variable precision DSP block of similar features is included in Cyclone V FPGA. Altera's latest MAX 10 FPGA incorporates embedded multiplier that can be configured as either one 18×18 multiplier or two 9×9 multiplier [101]. Supporting higher bit width multiplication is supported by cascading multiple embedded multiplier blocks together. Over the generations, the number of DSP blocks available has also increased significantly.

A comparison of DSP blocks used in Stratix series FPGAs are shown in the Table 2.4.

(ii) Xilinx DSP block

Early FPGA devices like the Virtex-II series consists of simple 18×18 multipliers to perform arithmetic computations that operates at a frequency of 150 MHz [102]. These multipliers offer significant performance improvements and area savings for many signal processing algorithms. In the Virtex-IV device, these multiplier blocks were extended to DSP blocks (DSP48), by adding a 48-bit adder/subtrator, with a maximum frequency of up to 500 MHz. The size of multiplier has increased from 18×18 bits to 25×18 bits in the Virtex-V device, with a maximum frequency of 550 MHz. Figure 2.7 shows a DSP48E1 DSP block incorporated in Virtex-VI device, includes a 25 bit pre-adder at the multiplier input as an additional functionality, generating output at 600 MHz [5]. In the latest 7 series FPGAs from Xilinx, the same DSP block is used across all the device families and are capable of running at a maximum operating frequency of 740 MHz. Additional functionalities such as pattern detection logic, cascade and feedback circuitry and SIMD mode area all incorporated into the latest DSP48E1. A more recent DSP48E2 [103] is included in the next generation Xilinx UltraScale architecture, offer wider data widths and a new functionality for the pre-adder, an additional input to the ALU and a dedicated XOR sub-block to perform wide XORs. Over the generation, the number of DSP blocks available have also increased significantly.

A broad comparison of Xilinx DSP48s slices [104] is shown in Table 2.5.

2.2.2 Academic DSP Architectures

Over the years, a wide range of DSP architectures have been proposed by various researchers, typically aiming at specific applications (signal processing) as well as addressing different challenges such as low operating power and high computation

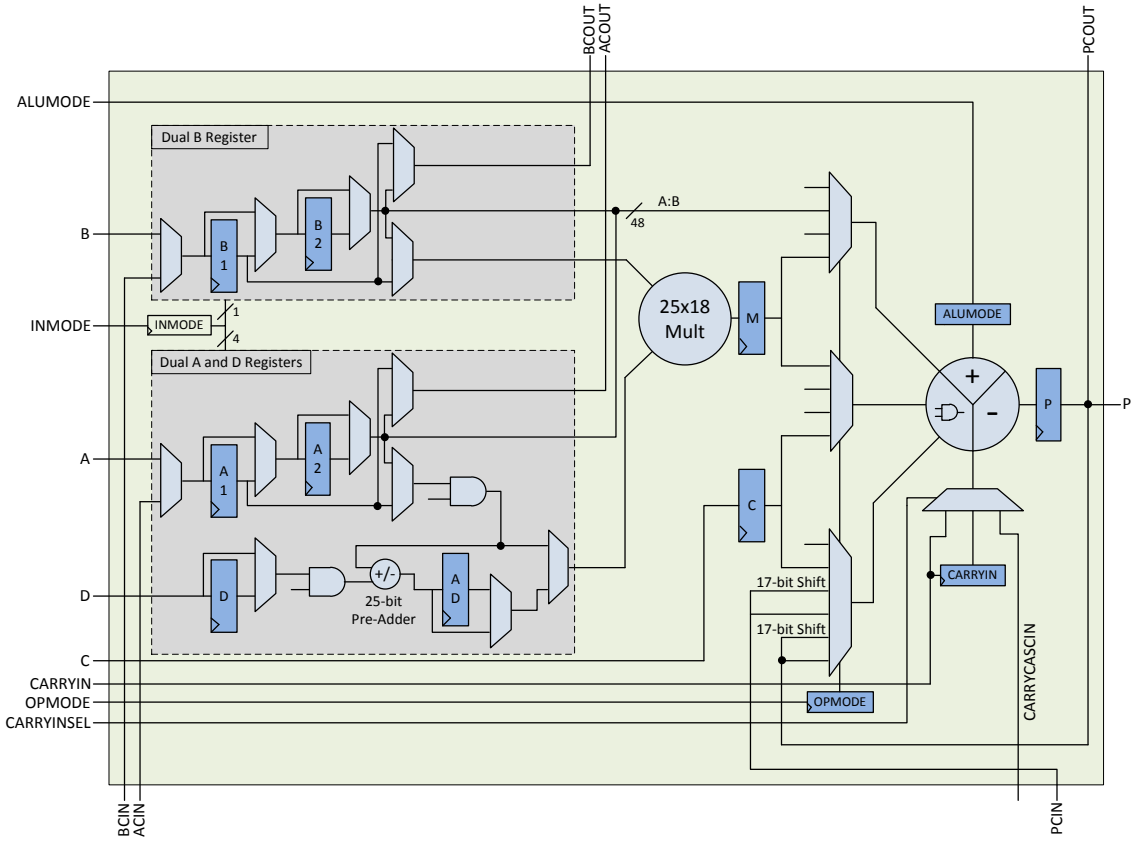


Figure 2.7: Detailed Block Diagram of DSP48E1 Primitive reproduced from [5].

performances. This section focuses on the academic DSP architectures and their advantages.

In [105], Kolagotla *et al.* proposed a high performance dual-MAC DSP architecture that augments the features of traditional DSPs and microcontrollers designed to give good performance when executing voice and video algorithms. The proposed DSP block is a modified Harvard architecture based processor with two 16-bit single cycle throughput multipliers, two 40-bit split data ALUs and hardware support for on-the-fly saturation and clipping to accelerate the fundamental operations associated with video and image based applications found in 3G algorithms. A novel scalable DSP architecture was proposed in [106] for SoC applications which can adapt according to the application. By employing a group memory, group registers and extra datapaths, the proposed expandable DSP architecture can be split into slices. By exploiting single instruction multiple data properties (SIMD) and

Table 2.5: DSP blocks evolution on Xilinx devices.

Devices	Capabilities	Speed	No of DSP Blocks
Virtex-2	18x18 bit multiplier	Up to 150 MHz	4-168
Virtex-4	18x18 bit multiplier 48-bit adder/subtractor (DSP48)	Up to 500 MHz	32-512
Virtex-5	18x25 bit multiplier 48-bit ALU (DSP48E)	Up to 550 MHz	32-1056
Virtex-6	25 bit pre-adder 18x25 bit multiplier 48-bit ALU (DSP48E1)	Up to 600 MHz	288-2016
Virtex-7	25 bit pre-adder 18x25 bit multiplier 48-bit ALU (DSP48E1)	Up to 740 MHz	60-2520
UltraScale	27 bit pre-adder 18x27 bit multiplier 48-bit ALU (DSP48E2)	Up to 740 MHz	600-5520

employing a modular Very Long Instruction Word (VLIW) Instruction Set Architecture (ISA), this scalable DSP architecture can be used for different system requirements.

Similar to Altera's variable precision DSP block, Parandeh *et al.* proposed a flexible DSP block which accelerates multi precision addition operations in addition to multiplication, thereby achieving flexibility [6]. The flexible DSP block integrates a by-passable partial product generator into a Field Programmable Compressor Tree (FPCT). This architecture enhancement allows the DSP block to be configured to perform multiplication, using FPCT for partial product reduction or to bypass the partial product generator to use FPCT for multi-bitwidth addition. The DSP block incorporates a 9×9 Baugh-Wooley (BW) multiplier. Figure 2.8, shows the flexible DSP block which contains two by-passable partial product generators and a half FPCT that can perform partial product reduction in conjunction with two hard logic compression circuits. However, when compared to a fixed function compressor trees, the area and power overhead is considerably larger. Hence the

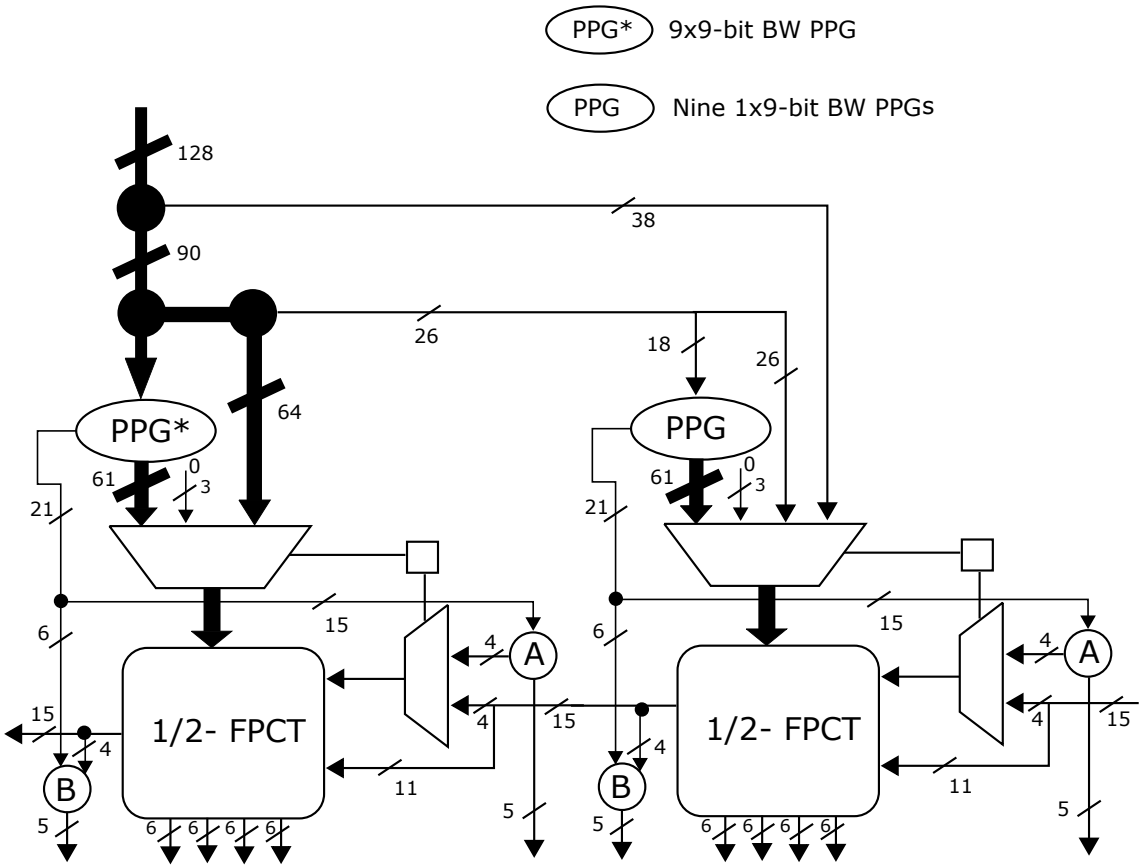


Figure 2.8: Flexible DSP architecture reproduced from [6].

proposed FPCT based DSP design makes it less suitable for low power implementations. An extension of the flexible DSP block was proposed by in [107], which performs variable precision multiplication in addition to multi-input addition without a considerable overhead. Using two paired 18×18 radix-4 Booth multipliers, the DSP block can perform 9/12/18/24/36-bit multiplications.

Apart from the academic DSP blocks architectures presented in the literature targeting the improved efficiency and performance, researchers also proposed special DSP architectures for specific applications. One such DSP architecture is the Signal processing On-Demand Architecture (SODA) which is a fully programmable DSP architecture that supports Software Defined Radio (SDR) [7]. One of the main characteristics of this processor is its support for different wireless protocols to make it suitable for SDR. The SODA multiprocessor architecture consists of one controller and four ultra wide Single Instruction Multiple Data (SIMD) processing

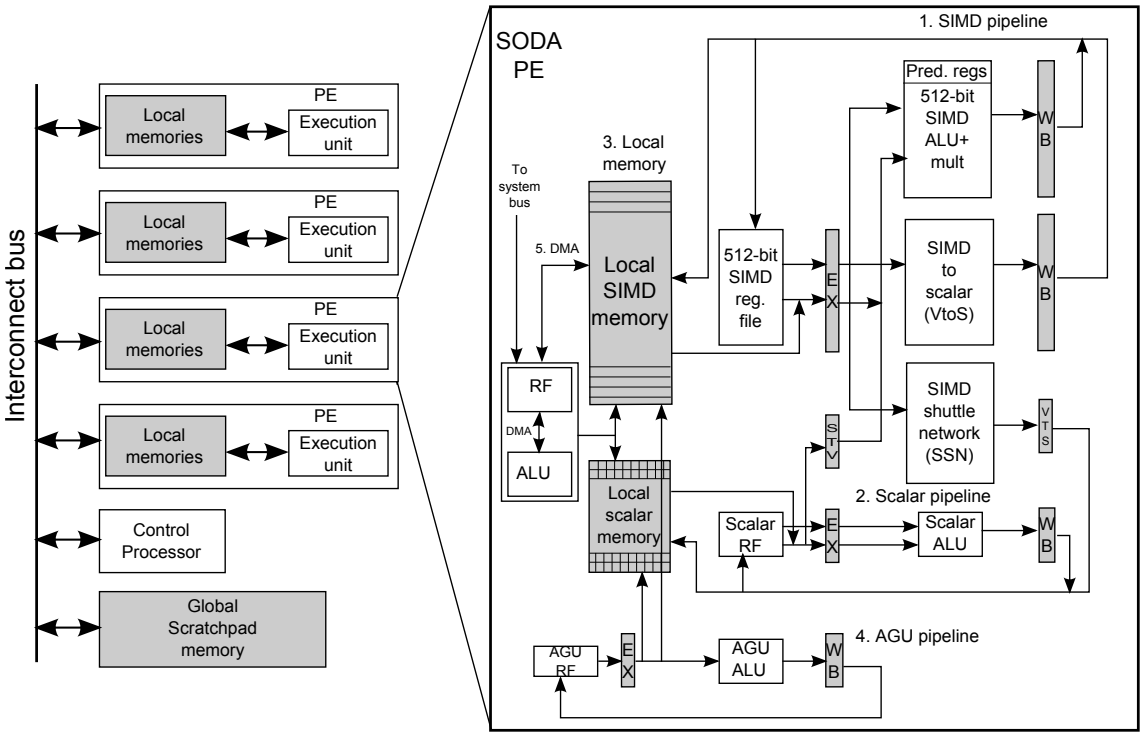


Figure 2.9: SODA architecture reproduced from [7].

elements with data communication done through explicit Direct Memory Access (DMA) instructions. The PE of SODA consists of five major components:

- A SIMD pipeline for supporting vector operations.
- A scalar pipeline for sequential operations.
- Two local scratch-pad memories for SIMD pipeline and the scalar pipeline.
- An Address Generation Unit (AGU) pipeline for providing the address for local memory access.
- A programmable DMA unit to transfer data between memories and interface with the outside system.

The SIMD pipeline that supports intra-processor data movements consists of a 32 way 16-bit integer datapath with 32 arithmetic units working in lock-step. Each data path includes two read-port, 16-entry register write-port, one 16-entry register file and one 16-bit ALU with multiplier. When SODA runs at 400 MHz,

the multiplier takes two execution cycles. SODA supports several SIMD reduction operations, including vector summation and finding its minimum and maximum. For current generation wireless systems, SODA meets the processing requirement of two widely used protocols (W-CDMA and 802.11a) within the strict power constraints of a mobile terminal. It is also designed to meet the demands of next generation wireless protocols through more processing elements with wider SIMD execution. The architecture of SODA is shown in the Figure 2.9.

2.3 Design Methodologies

Efficient and optimal mapping of input circuit designs on FPGA platform requires an equally efficient CAD tool. Implementing hardware on an FPGA generally begins with a behavioural description of the digital circuit in a hardware description language. The behavioural representation of the digital design has to be validated using an RTL simulator (e.g. Modelsim [108]) to ensure the correctness of the function. In the validation process, the *golden* output (desired output) is compared with simulated output obtained using representative test vectors. Manual debugging of an RTL description involves stepping through the signal waveforms at each clock cycle. Automated validation using checker and tracker modules are employed for large systems [108]. The designs that are validated to be functionally correct, but do not meet the timing or area constraints, have to be modified, revalidated and re-synthesized before the synthesize process. Synthesis converts the validated behavioural description into a netlist of basic circuit elements. Various technology-independent optimization techniques are applied on the generated network at this stage to minimize the number of logic gates. The output of a synthesis stage is a network of Boolean logic elements, flip-flops, basic circuits and corresponding connections between them. The following sections discuss on the technology mapping, placement and routing methods that follows synthesis step.

2.3.1 Technology Mapping

The aim of technology mapping is to map the given technology-independent description (output of synthesis step) to a specific technology while satisfying different cost metrics and constraints provided by the user. In technology mapping, the target application is mapped using a set of pre-defined logic and circuit components.

A typical input design contains both control path and datapath elements. The datapath elements like arithmetic operations and memory components are mapped onto dedicated embedded blocks such as DSP blocks and BRAMs. The network of control logic is translated into an interconnection of basic logic elements (LUTs and DFFs) of the underlying architecture. The library module of the architecture contains the LUT level expansions for control logic elements like multiplexers and bit wise operators. For a given input network of basic logic gates, technology mapping implements the circuit using a netlist of K-input LUTs. Logic elements with wider inputs (greater than K inputs) are realized using a cluster of LUTs. In some cases, logic duplication is also employed to optimize the mapping results.

Different techniques and algorithms are available in the literature for FPGA technology mapping. One of the techniques is tree based mapping [109] and its extended versions [110, 111], in which the input netlist is partitioned into tree structures and each tree is handled separately. Each tree structure is mapped for area minimization by enumerating all possible LUT implementations using dynamic programming. Another approach is the flow-based mapping technique (FlowMap) proposed by Cong *et al.* [112]. FlowMap is based on the max-flow-min-cut theorem and the network flow computation. Although FlowMap can generate depth-optimal solutions in polynomial time, it lacks flexibility due to its limited depth optimal min-cut solutions for each node. Furthermore, its cut-enumeration based technique explores many possible cuts for every mode and offers high flexibility and optimality which come with the cost of performance, when compared to tree-based or flow-based approaches [113]. In addition to area and delay optimization techniques, there are algorithms available in the literature to optimize power

consumption [114] and routability [115, 116]. Furthermore, since technology mapping is an intermediate step in the FPGA mapping flow, a better solution can be obtained by merging the technology mapping step with optimization techniques, which can be performed before or/and after the mapping step. A more detailed perspective of technology mapping can be found in [117, 118].

2.3.2 FPGA Placement and Routing

The next step after technology mapping is the physical synthesis which mainly consists of placement and routing. The placement step determines the best possible location of each logic block. In placement, the communicating logic blocks are placed as close as possible to reduce the routing length and delay. The placement algorithm in VPR proposed by Betz *et al.* is based on Simulated Annealing (SA) [119]. The basic operation in the SA process is defined as swapping of the contents of two logic blocks. A placer can be net congestion-drive or timing-drive. The former minimizes routing resource while the latter minimizes delay. Hence, the cost of a placement can be determined based on its total routing wirelength or routing delay. A high quality solutions are obtained using SA as it covers large search space at the cost of longer runtime. Another placement technique is the partitioning-based placement for FPGA, determines the placement solution at reduced runtime (compilation time) compared to VPR with slight degradation in quality of solution. This approach is widely adopted in ASIC [120].

The final step in the FPGA mapping flow is routing that determines the routing resources used to carry signals between logic elements. Since FPGAs have prefabricated routing resources, achieving 100% routability becomes a major challenge. Generally there are two types of FPGA routing: global routing and detailed routing [118]. Global router decides which routing channel and routing switches are to be used for each net. PathFinder [121] and VPR [119] routers are the most widely used global routing techniques in FPGA which are based on negotiation-based iterative routing for ASIC designs. The iteration starts with a minimum cost function resulting in over-congestion in some routing channels. However, the cost

function is tuned to increase the penalty of routing through congested resources. The process is iterated until all the congestion is removed.

After global routing, the detailed routing determines the exact wire segments and pins that a signal travels through. Segment allocator (SEGA) [122] and Coarse-Grain Expansion (CGE) detailed router [123] are the most widely used detailed routing algorithms. These types of algorithms generally perform three steps for each net: selecting the solution with lowest cost, removing alternative solutions and removing the solution that conflicts with selected net. Routing cost is calculated based on the number of alternative paths and resource usage. Nets that have limited alternative solutions are given higher priority to be processed earlier. Another detailed routing approach is Boolean satisfiability (SAT) formulation which performs detailed routing based on Boolean expressions in the conjunctive normal form. There are other FPGA routing algorithms available in the literature such as the negotiation based VPR router [124] that combines both global routing and detailed routing in one step.

2.4 FPGA Power Optimization

Traditionally, research work on FPGA has focused on addressing area and speed overheads [1]. However, with the increased demand for low power applications and technology scaling, much of the focus has shifted to improve energy efficient mapping on FPGAs. This section reviews the basic energy optimization schemes employed at different levels of the FPGA platform.

2.4.1 System Level Power Optimization

Low-power techniques at system level can be classified into three categories: basic techniques, runtime reconfigurability techniques, and techniques for soft processors. The basic techniques of power optimization include the mapping of circuits using coarse-grained blocks more than the fine-grained logic blocks, as the former

is more power efficient than the latter for the same function [125]. Pipelining is another basic implementation technique that reduces glitches, and hence minimize the power consumption [126]. Also, other basic power optimization schemes like clock gating and dynamic voltage scaling techniques are also available in the literature [127].

Low-power techniques involving runtime reconfigurability have been widely studied by researchers. Osborne *et al.* proposed a technique of combining word-length optimization with runtime reconfiguration so as to adopt the smallest design at a given time to minimize energy consumption [128]. Another system-level power optimization scheme is the FPGA-based soft processor. Instruction set extensions to the MicroBlaze soft processor result in 40% reduction in energy and 12% reduction in peak power [129]. Dynamic power reduction up to 74% has been reported by power aware scheduling and instruction re-ordering techniques to optimize a soft processor at multiple levels of abstraction [130].

2.4.2 Circuit and Architecture Level Power Optimization

One of the most efficient ways to reduce FPGA power consumption is the use of an optimized architecture and circuit-level implementation, as it directly affects the efficiency of mapping applications on FPGA resources, and the amount of circuitry needed to implement these resources. In [25], George *et al.* proposed both architectural and circuit-level optimization concurrently to obtain an energy efficient FPGA platform. It includes various hybrid interconnect structures like nearest neighbour connections, hierarchical connectivity and symmetric mesh architectures for reconfiguration, making it suitable for low-power embedded applications. A low swing signaling technique and hybrid switch solutions composed of both buffer and pass-gate switches without degrading the performance of FPGA is proposed in [131]. A new FPGA routing architecture that utilizes a mixture of hardwired and traditional flexible routing switches, which reduce the number of configurable routing elements resulting in 24% reduction in leakage power consumption is proposed [132]. A novel FPGA routing switch that can operate in

high-speed, low-power or sleep modes is proposed [133], which reduces the leakage and dynamic power without any router complexity. To reduce static power consumption, a power-gating technique is applied to the routing switches and dynamic power reduction is achieved by duplicating routing resources that use either high or low Vdd [134]. Power efficient modules for embedded components were introduced in [135]. These optimize the number of connections between the module and the routing resources, and use reduced supply voltage circuit techniques. Moreover, power reduction techniques like register file elimination and efficient instruction fetch for a coarse-grained FPGA cell based architecture are proposed in [136].

2.4.3 FPGA Mapping Tool Level Power Optimization

Over the years, researchers have also worked on mapping tool level optimization of FPGA implementation to evaluate the area-delay trade-offs. In [137], area-delay estimation is determined based on a library of benchmarks implemented and characterized for several FPGAs. The main bottleneck of this approach is the requirement of different libraries for different tasks for various devices and applications. Based on a Xilinx XC4000 device, Xu. *et al.* proposed a technique to compute area and delay values from an estimation obtained from mapping and place & route stages [138, 139]. The technique proposed in [140] estimates area and delay at the abstraction level (DFG) in which a combination of both algorithm characterization and FPGA mapping model is used. However, it fails to consider the control and multidimensional data overheads. Area-delay estimation based on compilation of high-level specifications and target loop transformations are presented in [141, 142, 143, 144]. In [141], a MATCH compiler is proposed to perform design space exploration in order to determine the area and delay after a set of operations: scheduling, register binding and interconnection delay. Computing area-delay values based on datapath estimation and memory control units are also considered in studies. An iterative compilation-based method is proposed in [142], which performs extensive transformation, approximation and optimization

of DFF nodes that are to be mapped onto FPGA to compute the best area and delay values. [144] is another compiler-based technique that estimates area and delay based on the synthesis results of a high-level synthesis tool. Furthermore, a very accurate area-delay estimation using analytical and empirical techniques is proposed in [145]. Choi *et al.* proposed a technique suitable for a large design space to efficiently estimate the area delay values [146].

Intelligent FPGA mapping tools are required to efficiently map an application to the FPGA programmable fabric. These mapping tools can also have a significant impact on power consumption. There are several stages involved in the FPGA CAD tools when mapping an application: high-level synthesis, technology mapping, clustering, placement and finally routing as discussed in section 2.1.1. Optimization of different stages results in power-efficient optimal mapping of the circuit design on to FPGA platform.

Power optimization in high-level synthesis for low-power FPGAs designs are presented in [147, 148, 149]. In [147], power consumption is reduced by minimizing the total wire length. Furthermore, fast switching activity and multiplexer optimization algorithms are applied to further reduce the power consumption of the design. In [149], an optimal algorithm is proposed in which the power is minimized by assigning the maximum number of operations to low Vdd and minimizing total switching activity through functional unit binding for the design. There have been extensive studies on power optimization in technology mapping for FPGAs designs [150, 151, 152, 114, 153, 154, 155]. The algorithms proposed in these works minimize FPGA power by minimizing high-activity nodes as much as possible when the logic gates are packed into LUTs and/or by minimizing node-duplication, which tends to increase the number of interconnects between the LUTs. Low-power clustering techniques have been widely studied by researchers [156, 157, 158]. In general, these algorithms minimize power by absorbing low fan-out and high-activity nets while clustering LUTs. The inter-cluster nets that dissipate most power are considerably reduced when small nets (low fan-out) are absorbed resulting in reduced power utilization. Power reduction techniques employed in place and route

stages are also available in the literature [159, 160, 161]. [160] employs probabilistic measures to determine the signal activities at the internal nodes of a circuit, based on which optimized placement and routing are performed to achieve low-power dissipation. The algorithm proposed in [151] minimizes leakage power by choosing low-leakage LUT configurations. Finally, power-aware algorithms for mapping logical memories to embedded memories in FPGA are proposed in [162]. The proposed technique evaluates different possible mapping options and chooses the most power efficient choice of embedded memories.

2.5 Summary

In this chapter, an overview of conventional FPGA architectures and various stages for mapping a design onto FPGAs are discussed. The main features and benefits of commercial and academic FPGA architectures are explained in detail. The significant characteristics of next generation reconfigurable architectures based on nanodevices are also presented. Hard blocks (DSP) widely adopted in the commercial world for realizing DSP-based applications are further discussed. Academic DSP designs which are developed for specific applications as well as to address generic challenges were explained further. Different algorithms used in technology mapping, placement and routing are also reported. Various studies in the literature that report the contributions of low-power FPGA implementation at different levels (system, circuit, architecture and tool) are also discussed.

The research reported in this thesis focuses on exploring DSP block design and its optimization for hybrid multi-context reconfigurable architecture that can efficiently realize arithmetic operations. In addition to the DSP architecture exploration, the thesis also looks into mapping tool optimization for hybrid multi-context architecture for an area/power aware mapping.

3

The NATURE Platform

This chapter introduces the background information on the fine-grained multi-context dynamic reconfigurable architecture, NATURE that is used as a testbed for the research work presented in this thesis. An overview of its hybrid reconfigurable architecture, its interconnectivity design and its corresponding mapping flow is also presented.

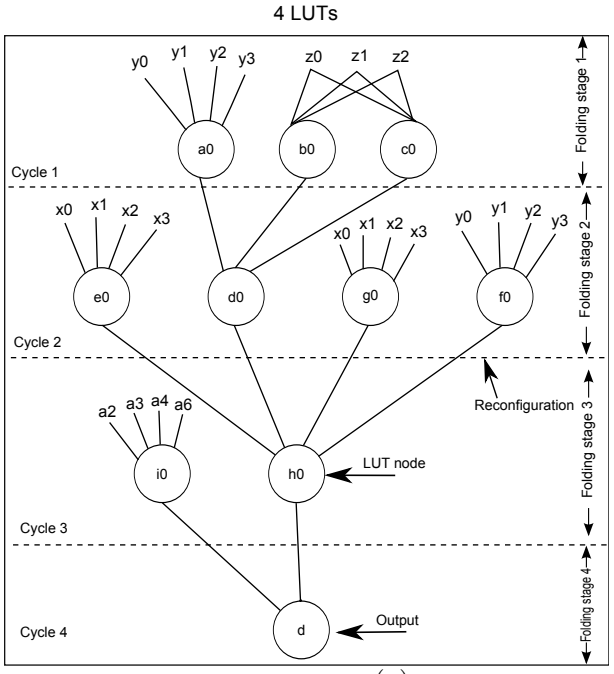
3.1 Dynamic Reconfigurable Architectures

Dynamic reconfiguration, also known as runtime reconfiguration performs re-allocation of hardware at runtime. Dynamic reconfiguration can improve the system performance by using highly-optimized circuits that are loaded and un-loaded dynamically during system operation. Multi-context reconfigurable platforms are one

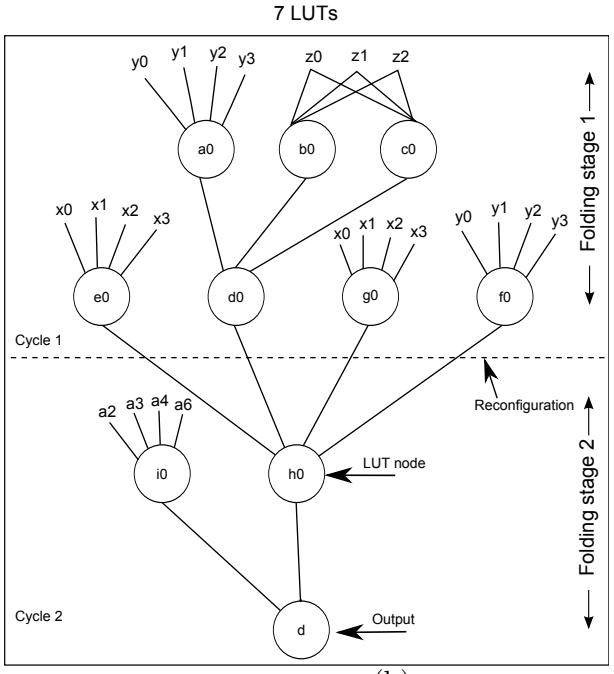
type of dynamic reconfiguration model in which configuration bits are stored in multiple configuration planes. NATURE is a fine-grained hybrid multi-context dynamic reconfigurable architecture that utilizes both CMOS logic and nano RAMs. It takes advantage of both technologies and thereby greatly improves the logic density and performance of design. An overview of its architectures is discussed in the following sections.

3.1.1 NATURE Architecture

The **NATURE** architecture is the acronym for CMOS/**NA**no**T**echnology reconfig**UR**abl**E** architecture proposed by Zhang et.al [163], which can be fabricated with CMOS-fabrication compatible manufacturing processes and leverage the benefits of both technologies. In this architecture, the highly-dense nonvolatile nano RAMs are distributed throughout the SoC design to allow large embedded on-chip configuration storage, which enables fast reading and hence supports fine-grained runtime reconfiguration. NATURE's Logic Elements (LEs) and interconnects can be reconfigured every few cycles, or even every cycle, making it a fine-grained runtime reconfigurable architecture. Compared to NATURE, conventional architectures allow only partial dynamic reconfiguration [164], in which only a part of the architecture can be reconfigured at runtime. In this new hybrid architecture, nano RAMs can be used to store a large set of configuration bits with a smaller area overhead. The nano RAMs can be Nonvolatile RAMs (NRAMs) [22], Phase Change Memory (PCMs) [23], Magnetoresistive RAMs (MRAMs) [24] or other such RAMs, which are nonvolatile, high density, high speed and low power. The distributed nonvolatile nano RAMs [24] and two SRAM cells per switch (one cell configured for the current computation while a second cell gets configured for the successive computation) significantly reduces the reconfiguration delay. As a result, NATURE architecture is able to provide a fine-grained cycle-level reconfiguration capability.



(a)



(b)

Figure 3.1: (a) Level-1 folding. (b) Level-2 folding.

Temporal Logic folding

Temporal logic folding is the basic concept used in the NATURE architecture, which is similar to the temporal pipelining used in DPGA [75]. Temporal logic folding is the ability to partition the circuit into a cascade of stages, which are

then implemented using the same set of logic elements through very fast dynamic reconfiguration. Hence a significant improvement in logic density and area can be achieved. Logic folding can have different levels of granularity, resulting in different area or delay characteristics. This provide significant flexibility in performing area-delay trade-offs. Figure 3.1a illustrates the concept of folding used in NATURE, in which each node denotes a LUT and the connectivity between them is represented by lines. Using the conventional FPGA, *10* LUTs will be required to realize this LUT graph shown in Figure 3.1a. However with NATURE for level-1 folding shown in the Figure 3.1a, only 4 LUTs are needed as each LUT can be reconfigured after every single level of LUT execution. In level-2 folding shown in Figure 3.1b, the LUTs are reconfigured after two levels of LUT execution, consuming a maximum of 7 LUTs.

The current library module of NATURE supports four folding levels which are folding levels 1, 2, 4 and 0. There are different trade-offs involved in the choice of folding level. As the folding level varies, the area and delay varies accordingly. For a given circuit, changing the folding level from 1 to 2, 4 and 0 leads to a higher clock period, but the cycle count reduces, since a larger number of LUT computations can be performed within a single clock cycle. This is because more operations are performed within a single clock cycle. Since the circuit execution is synchronized, the total circuit delay equals the clock period times the total number of clock cycles. Typically overall circuit delay decreases with increase in folding level. If area is a constraint for implementing a logic circuit, then a small folding level is usually a better choice as area utilization decreases.

Architecture of NATURE

Like conventional FPGA architectures, NATURE also has an island style architectural design as shown in Figure 3.2. The basic reconfigurable block used in NATURE is called a Logic Block (LB) which contains a Super Macro-Block (SMB) and a switch matrix connected by various levels of interconnect, supporting local and global communications among LBs. The switch matrix acts as an interface

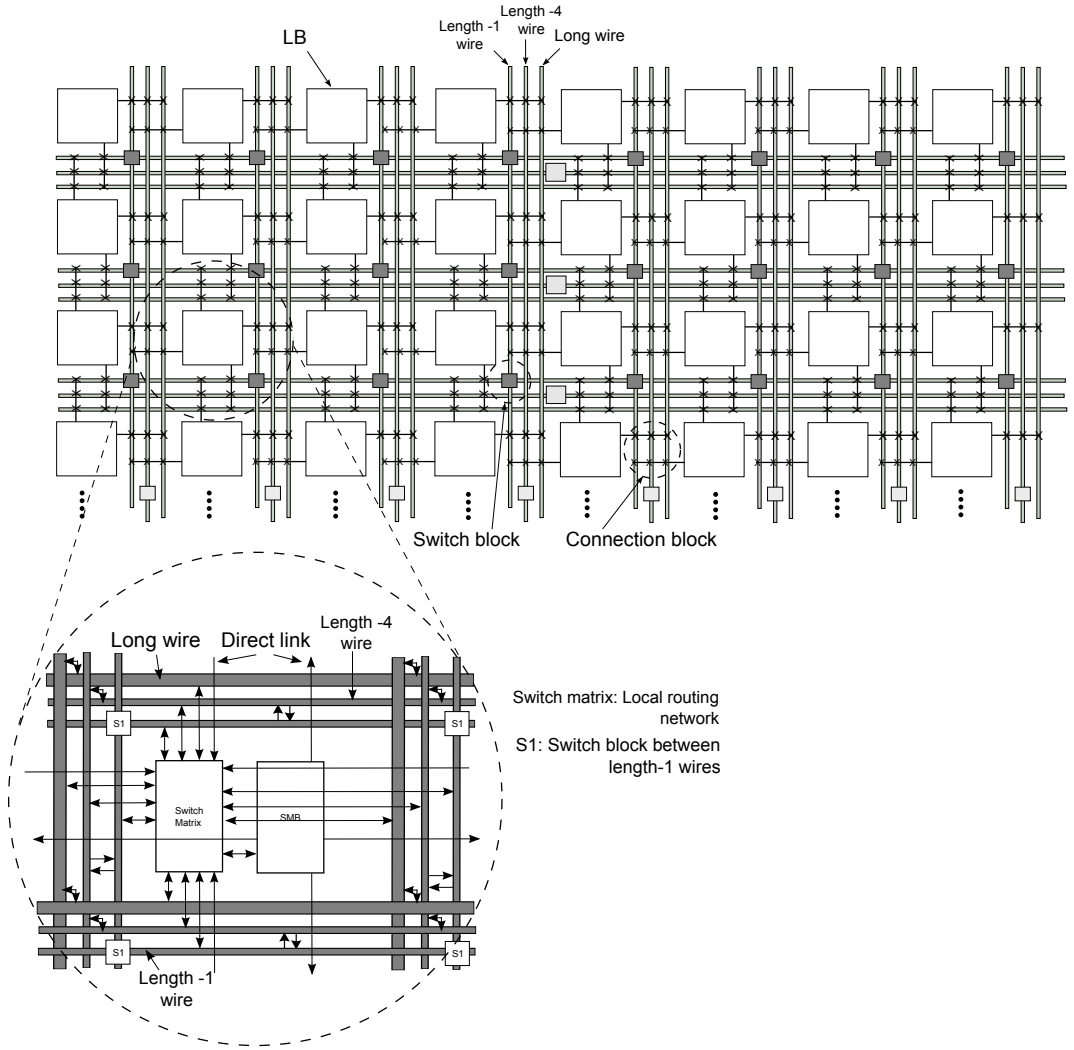


Figure 3.2: Architecture of NATURE reproduced from [8].

between input and output edges of the SMB. Interconnection between adjacent SMBs are done through direct links. The SMB in NATURE architecture contains two levels of logic to support temporal logic folding. In this hierarchical architecture as shown in the Figure 3.3, one SMB comprises 4 Macro Blocks (MBs) in the first (i.e higher) level, and 4 LEs per MB in the second level, resulting in 16 LEs per SMB.

The communication between MBs in an SMB is through local crossbar or MUX. Full connectivity is provided among various components inside MB and SMB. As shown in the Figure 3.4, an LE contains one *4*- input LUT and 2 FFs. The FF can be used to store the computation result of the LUT for future use or to store some intermediate results from the LUT output.

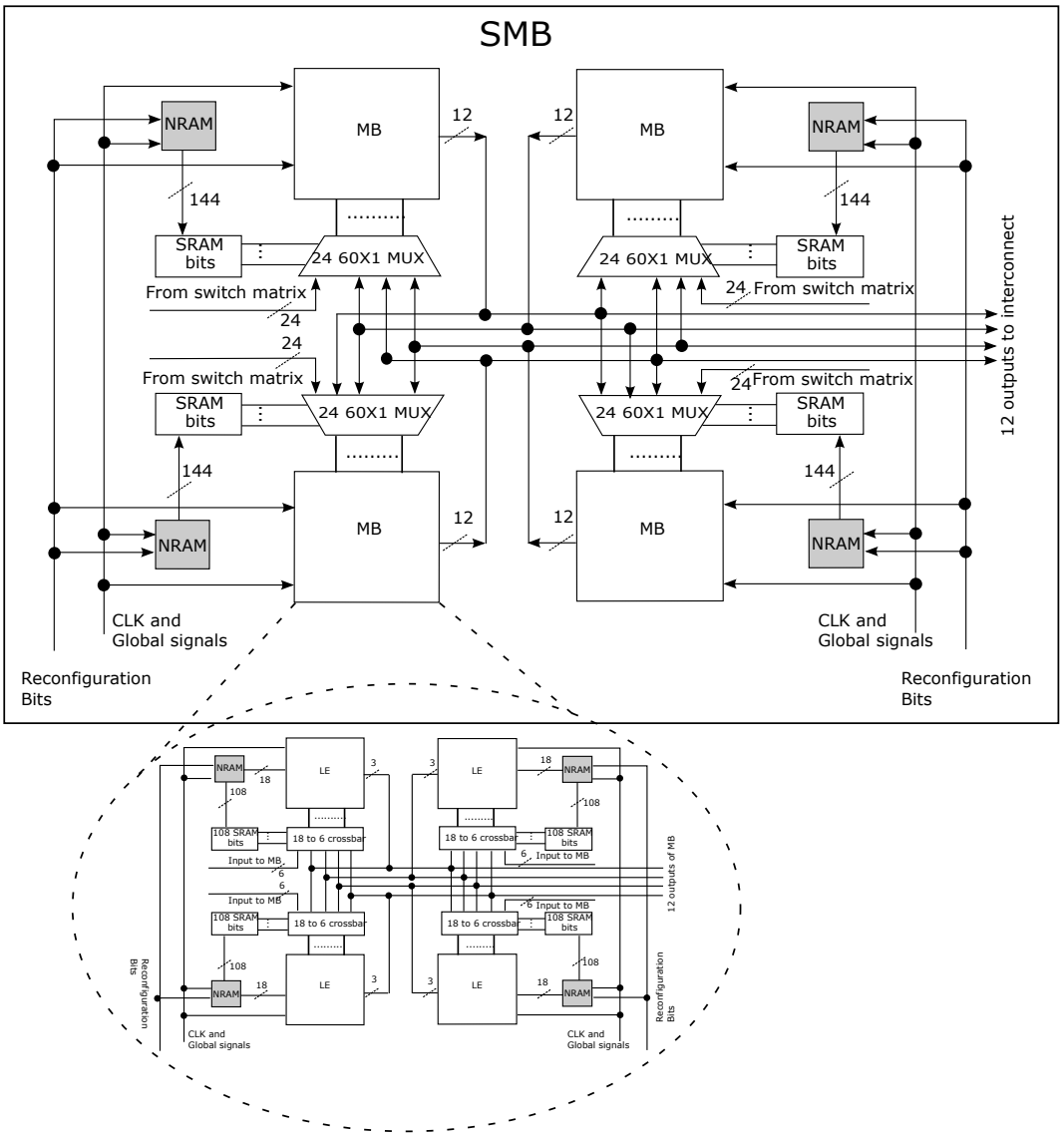


Figure 3.3: Hierarchical architecture of SMB reproduced from [8].

The routing architecture of NATURE is mainly island style. Within the SMB, the interconnect is hierarchical to aid logic clusters and local communication. Wire segments of various lengths are used to connect SMBs. NATURE routing architecture employs mixed wire segment schemes including length-1, length-4 and long wires, for routing flexibility and circuit delay control. There are also direct links from the output of LB to its four neighbouring LBs to further facilitate local communications. The routing structure caters for all levels of folding, including the no-folding case. When employing logic folding, all the interconnects and switches can be reconfigured by their associated nano RAMs and reused every cycle, which reduces global communication significantly.

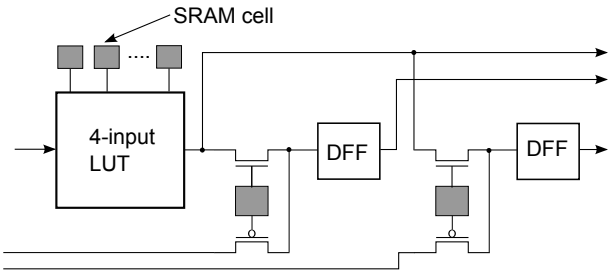


Figure 3.4: Architecture of an LE with 4-input LUT and two flip-flops reproduced from [8].

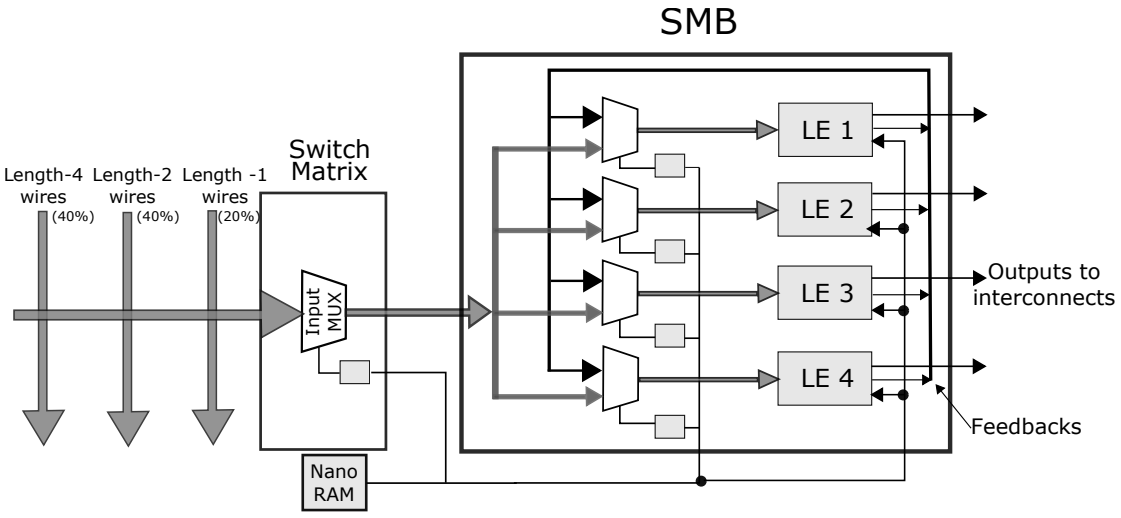


Figure 3.5: Flattened NATURE architecture reproduced from [9].

The NATURE architecture can be flattened such that there is only one level of logic (one layer) in which an SMB contains only 4 LEs as shown in the Figure 3.5. The inputs of LE can be selected from the common inputs of the SMB and the feedbacks. The common inputs, which are part of the total total number of inputs needed, serve as the inputs to all the LEs in an SMB. To minimize the interconnect delay while connecting LEs, 70% of the primary inputs are shared as common inputs in the flattened design. Large multiplexer clusters are incorporated in the flattened architecture to facilitate LE interconnection. The flattened NATURE architecture facilitates local communication within the SMB; however, it increases the delay from outside of the SMB to the inside of an LE. The flattened NATURE architecture uses a mixed wire segment scheme including 20%, 40% and 40% for length-1, length-2 and length-4 wires segments respectively. This thesis mainly focuses on the flattened NATURE architecture for efficient mapping of compute intensive arithmetic operations.

3.2 Mapping Tools For Dynamic Reconfigurable Architectures

A mapping tool is an integrated platform that takes an application design described in RTL and converts it to a stream of bits that is eventually programmed on the FPGA. In this section, the design optimization mapping tool developed for NATURE, the NanoMap tool is briefly introduced.

3.2.1 NanoMap Tool Flow

NanoMap is an integrated mapping platform that was developed to be used for NATURE. It performs design optimization from the RTL level down to physical level. Given an input circuit specification in RTL or in gate-level HDL after logic synthesis, the tool chain optimizes and implements the design on NATURE through logic mapping, temporal clustering, placement and routing. NanoMap automatically identifies the best temporal logic folding configuration, targeting area, delay or area-delay product. It uses a Force Directed Scheduling (FDS) [165] technique to optimize and balance resource usage across different cycles. Figure 3.6 shows the complete flow of NanoMap. The main processing steps of NanoMap are explained below:

(i) Logic Mapping

In this step, the user-specified constraints, the folding level is determined. Based on the folding level, RTL modules are partitioned into connected LUT clusters, followed by scheduling of LUT and LUT clusters into appropriate folding stages. Various sub-steps involved in logic mapping are further discussed below.

(i-a) Plane Search: Plane search uses an iterative approach to identify the best folding level based on different parameters such as user-specified design constraints,

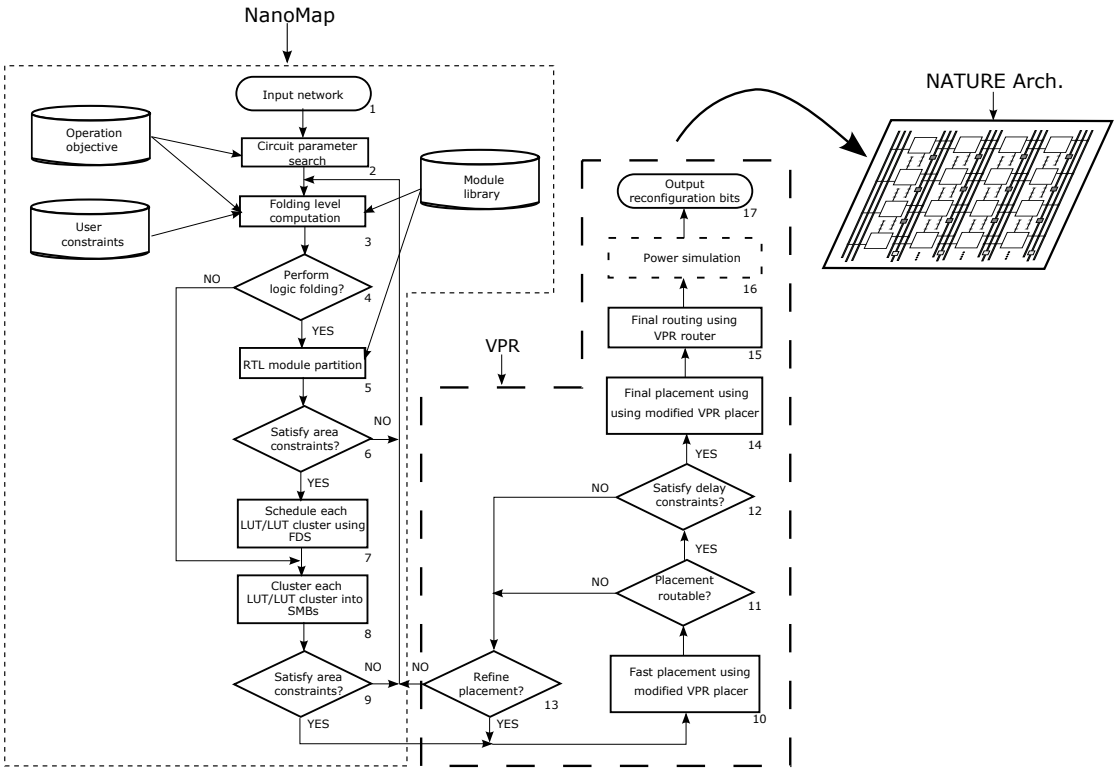


Figure 3.6: NanoMap design flow reproduced from [10].

optimization objectives, and input circuit structure. Logic between two levels of registers is referred to as a *plane*. Plane separation is the key part of the circuit parameter search, which collects the circuit parameters for choosing the folding level. To identify each plane, a depth-first search algorithm is used. The algorithm follows the structure by following the edges of each node and assign the nodes (modules or LUT) along its path into the current plane. A new plane is created when the algorithm encounter registers associated with the plane during its extensive search.

(i-b) Folding Level Computation: This is the second step in logic mapping. Based on the input circuit parameters and user constraints, the initial folding level is computed but is further adjusted as needed during each iteration of NanoMap till an optimized folding level is obtained. Folding level selection is critical to achieve best area-delay trade-off for a specific optimization objective. The factors that are used to calculate the folding level are:

- Total number of planes.
- LUTs per plane.
- Maximum number of LUTs among all the planes.
- Width of each plane.
- Maximum plane width among all the planes.
- Logic depth of each plane.
- Maximum logic depth among all planes.
- Area constraint (Maximum number of LUTs available).
- Delay constraint.
- Number of reconfiguration copies in each nano RAM.

The folding level is calculated based on the different optimization objectives and constraints.

For delay optimization with no area constraint, the process starts with no folding and then iteratively searches for the optimal folding levels to obtain shortest delay. If the area constraint is given, then it has to be met first, followed by fitting the circuit to the best possible delay. Two scenarios have to be taken into consideration:

- Resource sharing across multiple planes: This scenario is realizable if there is no feedback across planes. Here the planes are stacked together so that the resources can be shared which reduces the area without affecting the delay. However if the maximum number of LUT across the plane is greater than available resources, then each plane is folded internally to further reduce area. For a balanced logic, the initial maximum folding level is given by

$$\#initial\_level\_max = \left\lceil \frac{depth\_max}{folding\_stage\_min} \right\rceil \tag{3.1}$$

As most of the circuit logic is not balanced along critical path, a minimum initial folding level using maximum plane width is obtained as below:

$$\#initial\_level\_min = \left\lceil \frac{available\_LE}{width\_max} \right\rceil \tag{3.2}$$

Next, the minimum folding level *min_level*, allowed by *num_reconf:*

$$min\_level = \left\lceil \frac{depth\_max * num\_plane}{num\_reconf} \right\rceil \tag{3.3}$$

The final minimum initial folding is given by:

$$\#initial\_level\_min = max\{min\_level, initial\_level\_min\} \tag{3.4}$$

Using the obtained folding level, NanoMap performs scheduling and clustering to obtain the area. If the area constraint is not satisfied, the folding level is reduced by one. Otherwise the folding level is increased by one and can be verified if the area constraint is still satisfied and delay is reduced.

- Multiple planes without resource sharing: If the circuit is pipelined, then these pipeline stages have to co-exist in FPGA and hence the scope of logic folding is limited within each plane. The parameters in this scenario for folding level calculation become $\sum_i num\_LUT_i$ or $\sum_i width\_plane_i$

If the objective is to minimize the area without timing constraint, then logic folding is performed as much as possible. Hence, within the plane, the minimum folding level value should be used. If a timing constraint is specified, the pipeline stage delay of the circuit should be less than the specified time constraint to maintain the required throughput.

(i-c) Force-Directed Scheduling (FDS): Scheduling is the process of deciding how to commit resources between a variety of possible tasks [166]. In a multitasking and multiprocessing operating system, scheduling allocates time slots for

execution, and in high-level synthesis, scheduling assigns data path operations to the best control steps to achieve desired cost-speed trade-offs. In the NanoMap tool, FDS algorithm is incorporated as the algorithm is intended to reduce the hardware resource usage, such as functional units, registers and bus required, by balancing the concurrency of the operations assigned to them without lengthening the execution time. FDS uses an iterative approach to obtain the schedule of operations in order to minimize overall resources usage [165], where the resources are modeled as force. Scheduling of an operation to some time slot, which results in minimum force, indicates a minimum increase in resource usage. Force is calculated based on Distribution Graph (DG), which describes the probability of resource usage for a type of operation in each time slot. FDS scheduling is used to assign LUT or LUT clusters to folding stages and balance the resource usage of folding stages. Once the force of each LUT or LUT clusters is calculated based on the LUT computation and register storage DGs, the node with minimum force is chosen and assigned to the folding cycle with minimum force. This will lead to a minimal increase in resource usage. This is an iterative process and it continues until all LUTs or LUT cluster computations are scheduled.

(ii) Temporal Clustering

By the end of scheduling, nodes are assigned to each folding cycle. In temporal clustering, the assigned LUTs are grouped into LEs and the LEs are packed into MBs and SMBs using a constructive algorithm. To populate each SMB, a LUT cluster with maximum number of inputs is selected and LUT with maximum number of its inputs within the cluster is selected as initial seed. A new LUT with high attraction to the seed LUT is chosen and assigned to the SMB, until the SMB is fully packed. Then a new LUT seed is selected.

Due to logic folding, several folding stages may be mapped to a set of LEs, some LEs may be used to hold the internal results and transfer to another folding cycle. In temporal clustering, both the LUT mapping and the corresponding registers storage life time have to be carefully tracked.

(iii) Temporal Placement and Routing

VPR is a placement and routing tool for array-based FPGAs [167, 124]. It allows the circuits to be placed and routed on a wide variety of FPGAs to facilitate comparisons of different architectures. It takes two input files: a netlist describing the circuit to be placed and routed and a description of the FPGA architecture. A modified VPR tool is used for placement and routing on the NATURE architecture. To evaluate the quality of initial placement, VPR's routability delay analysis is used, which determines whether a high-precision placement is achieved, or another round of optimization using adjusted logic folding should be involved. The algorithm used for placement is simulated annealing. To provide support for the inter-folding stages, the VPR placer has been modified. Once the placement is over, detailed routing can be performed using VPR to connect all SMBs in each folding stage and finish the mapping process. Direct link capability is included in the router to support the interconnect structure in NATURE. The routing in VPR can be conducted in a timing driven-fashion. As a result, the router tends to use the wire segments in a systematic fashion to attain the timing constraints. Based on the placement and routing on the NATURE architecture, the reconfiguration bits for each LE/switch can be generated.

3.3 Summary

In this chapter, a detailed overview of fine-grained NATURE architecture design and its routing structure is presented. The flattened NATURE architecture, on which the thesis is based, is discussed. The chapter also described different steps involved in the NanoMap tool flow for efficient mapping of circuits on the NATURE architecture.

4

Full-Block Reconfigurable DSP for NATURE

4.1 Introduction

FPGAs can be categorized as either fine-grained or coarse-grained [168] architectures. In fine-grained FPGA architectures, the logic block contains only basic logic units, consisting of one or two LUTs and groups of registers, implementing a single function on a small number of bits [11]. Coarse-grained FPGA architectures on the other hand are typically much larger, consisting of relatively complicated and optimized logic blocks designed for specific functions [169]. Modern FPGA designs tend to incorporate both fine-grained logic fabric and coarse-grained blocks. Such

mixed-grained architectures provide better design flexibility and performance for data-intensive applications.

The existing multi-context dynamic reconfiguration architecture described in the section 3.1.1 uses fine-grained logic blocks to realize various circuits. Implementing applications containing complex arithmetic operations using fine-grained logic affects the performance (in terms of speed) of the circuit. Hence, in order to reduce the delay, it is beneficial to implement complex arithmetic computations like extended multiplications and MAC operations in a coarse-grained block (like DSP) while implementing simple operations like control logic using fine-grained elements. This chapter proposes a novel full-block reconfigurable coarse-grained DSP for performing complex computations and demonstrate how the performance of current architecture can be improved.

The work presented in this chapter is also discussed in,

> R. Warrier, L. Hao, W. Zhang, *Reconfigurable DSP Block Design for Dynamically Reconfigurable Architecture*, in Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), pp. 2551-2554, June 2014.

4.2 Related Works

Over the years, a wide range of works on both commercial and academic DSP architectures for FPGAs have been reported in the literature. For example, Virtex-II FPGA family developed by Xilinx for high performance from low-density to high-density applications, incorporates a dedicated 18×18 multiplier block [102]. These multiplier blocks can be used to realize read/multiply/accumulate operations and implementing DSP filter structures efficiently up to a maximum operating frequency of 150MHz. Evolving from the 18×18 multiplier block, Xilinx has also developed specialized DSP blocks like DSP48E1 [170] with 18×25 multiplier that can operate at a maximum frequency of 740 MHz to efficiently perform complex signal processing computations. Altera also incorporates advanced DSP blocks in their

latest FPGA architecture which can support variable precision operations [171]. Apart from these commercially available architectures, DSP blocks like SODA [7] have also been designed for specific applications in academia. Furthermore, there are DSP block architectures from academic research with feature enhancements for supporting varied bit-width multiplications [107]. While some of the DSP blocks allow their functionality to be altered dynamically (dynamic programmability on Xilinx DSP48 series), vendor tools are currently unable to analyze the computational patterns and automatically reuse these DSP blocks through cycle-by-cycle reconfiguration. On Xilinx FPGAs, designers have to explicitly manage these reconfiguration operations by controlling the *INMODE, OPMODE* and *ALUMODE* configuration pins using state machine (or other similar logic), which incurs additional fine-grained resources like LUTs and FFs.

Multiply-and-accumulate unit (MAC) is the prevalent building unit in a DSP block. A generic MAC architecture consists of a multiplier, whose output is fed to a simple accumulator, adder/subtracter or a complex ALU unit, which supports post addition, subtraction or other bit-wise operations on the multiplier output to deliver the final results [172, 173]. Typically, a multiplier unit is built up by a sequence of partial product units and a carry propagation adder unit. The design of the multiplier unit is the major bottleneck that determines the performance of DSP block. Over the years, a wide range of multiplier algorithms and architecture optimizations have been developed to reduce the multiplier delay. One of the earliest multiplier designs is the Wallace tree multiplier [174]. A widely used multiplier implementation is the Modified-Booth Algorithm (MBA) together with a logarithmic depth reduction tree [175] and a final adder. Radix-4 MBA and parallel architecture MBA [176] are later introduced for high-speed multiplication. However, due to its high power dissipation, the MBA multiplier is not a good choice for implementing a low power MAC unit. Another popular multiplier algorithm is Baugh-Wooley (BW) [173] which is more power and energy efficient than the MBA multiplier of equal bit-width. Another way to increase the multiplier's performance on throughput is to minimize the critical path delay by inserting an extra pipeline register, either inside the PP unit or between the PP unit and final adder, leading

to a 3-cycle MAC architecture. However, adding an extra pipeline unit incurs overhead in terms of area and power and also causes an increase in the latency.

In this chapter, a runtime full-block reconfigurable DSP block which can support temporal logic folding is proposed and presented. In the full-block configuration, the DSP block can be reconfigured to implement different arithmetic operations **only after one full DSP operation cycle**. The internal interconnects of the proposed DSP block are designed to flexibly utilize the different functional units of our DSP block. The efficiency and flexibility of the proposed DSP is validated by implementing compute kernels such as wide multiplication and filter designs. The chapter also discusses the extension of NanoMap mapping tool flow to efficiently incorporate the proposed full-block reconfigurable DSP block into NATURE architecture.

4.3 Full-block Reconfigurable DSP for NATURE

This section first presents a motivational example to demonstrate the need to incorporate DSP in the existing NATURE architecture. Further sections explain the detailed design features of the proposed full-block DSP architecture, its reconfiguration capability and its interconnect design details. Finally the functionality of the proposed DSP block for supporting wider multiplication and filter designs are also discussed.

4.3.1 Motivational Example

The architecture of existing multi-context dynamic reconfigurable FPGA, NATURE is based on CMOS logic and nano RAMs. The use of on-chip nano RAMs allow large embedded on-chip configuration storage, which enables fast reading and hence supports fast fine-grained runtime reconfiguration. The concept of temporal logic folding in NATURE supports the folding of the logic circuit in time and maps each fold to the same fine-grained elements (LUTs and DFFs) in the architecture.

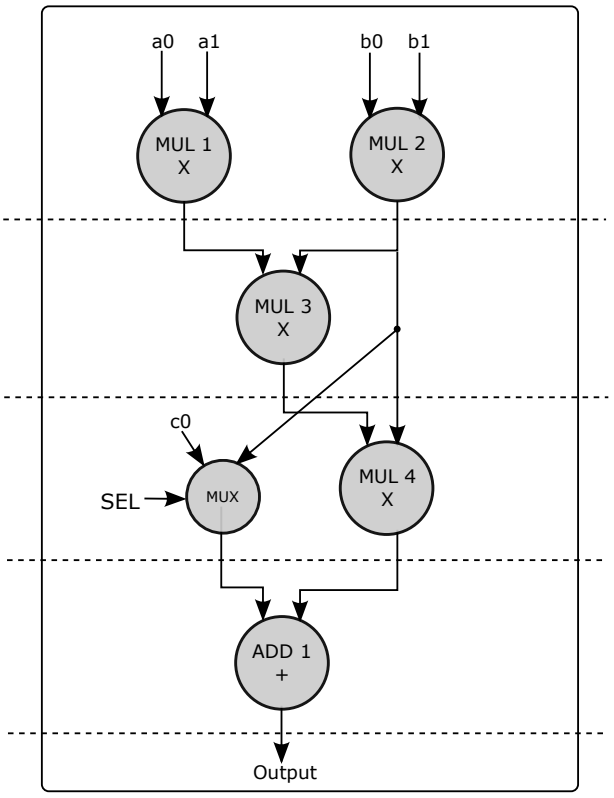


Figure 4.1: DFG illustration.

Although, the NATURE architecture improves logic density and area-delay product, the use of fine-grained logic limits the performance while mapping compute intensive kernels. This can be explained with a scheduled datapath illustrated in Figure 4.1, which consists of 4 multiplier nodes, 1 adder nodes and one multiplexer node. Realizing a 16 bit multiplier node using the existing flattened NATURE platform would consume 494 LUTs (124 SMBs) and 25 LUT computation cycles. Thus, the mapping of such arithmetic intensive nodes that lies along the critical path using NATURE architecture affects the performance. Also, more resources (LUTs and DFFs) are consumed resulting in a large area utilization for realizing such arithmetic intensive applications. To improve the performance of these operations, conventional FPGAs have integrated hard blocks, like the DSP block, which can perform these operations efficiently. The DSP block in Altera devices do not support fast runtime reconfiguration and cannot be reused during execution to increase efficiency. Although the DSP48E1 blocks on Xilinx FPGAs support dynamic programmability, user designs have to manually exploit them using additional logic to drive the configuration pins on a per-cycle basis. Also,

the vendor tools are currently unable to analyze a generic compute kernel and automatically reuse the DSP blocks during the implementation phase. Hence, realizing the DFG shown in the Figure 4.1 using Altera or Xilinx FPGA takes 4 DSP blocks; 4 separate DSP blocks for MUL 1, MUL 2, MUL 3 and the chained MUL 4 and ADD 1 nodes. This results in an increase in DSP block utilization while mapping generic compute kernels on commercial FPGA architectures, impacting area and power consumption. Also, the single pre-adder unit in the Altera and Xilinx DSP blocks cannot efficiently map certain class of operations like complex number multiplications.

This chapter introduces a full-block reconfigurable architecture that realizes arithmetic nodes in minimum LUT computation cycles thereby achieving better performance over fine-grained implementations. The proposed runtime reconfigurable DSP block exploits the temporal logic folding of NATURE architecture and facilitates the implementation of complex functions. The dynamic reconfigurability of the proposed DSP block allows its efficient reuse resulting in minimum resource utilization and improved power consumption. Finally, the DSP block is also incorporated with dual pre-adders for efficient mapping of POS based circuits. The detailed architecture of the full-block reconfigurable DSP block is discussed in the next section.

4.3.2 Proposed DSP Architecture

Implementation of compute intensive mathematical operations such as Finite Impulse Response (FIR) filters, Infinite Impulse Response (IIR) filters, Direct Cosine Transform (DCT) and Fast Fourier Transform (FFT) requires DSP as the basic building block. Dedicated DSP blocks are specifically designed to perform arithmetic operations, such as pre-add and multiply, multiply-add, and multiply-accumulate.

Commercial FPGAs from Xilinx and Altera support error checking and correction in primitives like BRAMs (using 1-bit parity per byte), resulting in non-standard

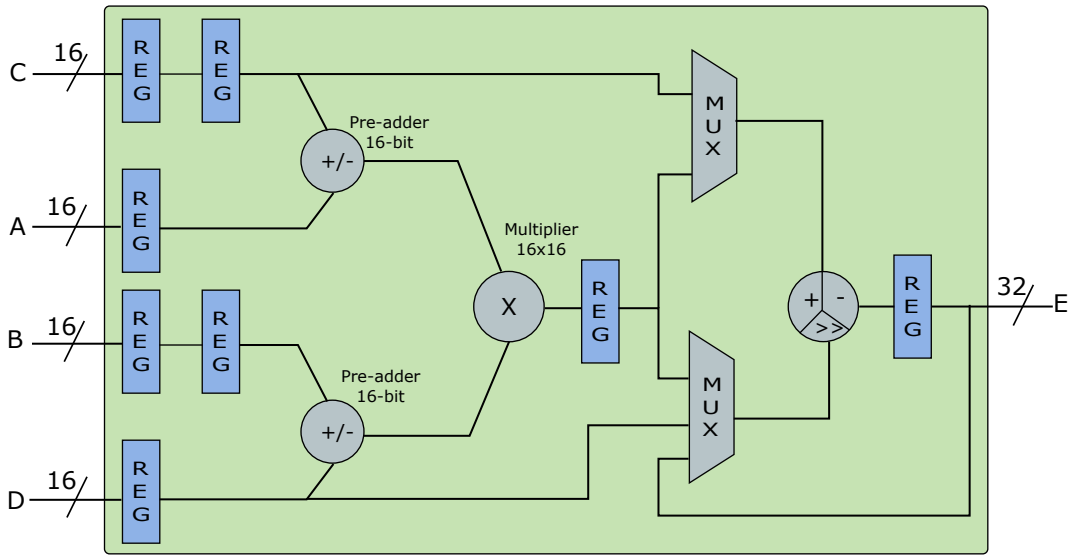


Figure 4.2: Basic structure of proposed full-block DSP.

interface width like 18-bit or 36-bit. The 18-bit input width of DSP/multiplier blocks in Altera and Xilinx devices enables direct interfacing to BRAM primitives, allowing efficient implementation of common circuits like filters [177, 178]. NATURE architecture uses unified 16-bit width for the BRAM modules and do not support parity bits. Similar to the case with commercial FPGAs, the bit-width of the multiplier is chosen as 16 bits to enable seamless integration with NATURE's BRAM modules, and library components, which support a maximum operand width of 16-bits.

The basic structure of the proposed full-block reconfigurable DSP design is shown in the Figure. 4.2. The DSP block consists of three main units: pre-adder, 16-bit multiplier, and arithmetic logic unit (ALU). Starting from the pre-adder unit, the output of pre-adders are connected to a 16×16 Wallace tree multiplier. The ALU unit is 32-bit wide and can perform various operations on the output of the multiplier and the other inputs. In contrast to the traditional DSP architecture in [5], the proposed DSP block has two pre-adders that support complex number multiplications and POS operations. There are pipeline registers incorporated in the DSP to enhance throughput. In addition, there are dedicated connections to support cascading multiple DSP blocks without using general routing resources. One of the most important features of the DSP block is that its functionality can be altered every few clock cycles through physical reconfiguration (configuration

bits stored in NRAMs), allowing it to perform different functions without requiring additional circuitry.

4.3.3 Detailed Design

This section describes in detail the internal connectivity of the different units, how the input signals pass through various pipeline stages and different multiplexer (MUX) configurations to select the appropriate signals for computations. A detailed architecture of the proposed DSP is shown in the Figure. 4.3. The DSP block has four input ports - C, A, B, and D respectively of uniform bit width. The MUXs C_port_Mux and B_port_Mux provide the flexibility to tap the signal from different stages of the input pipeline. The input signals through these ports are fed to the pre-adder input. The pre-adder is a 16-bit two input adder/subtracter. The DSP block is equipped with a 16×16 Wallace tree multiplier unit. Based on the MUX configuration at the input of the multiplier, it performs multiplication of the two pre-adder outputs, or the two input signals from ports A and B or a pre-adder output and an input signal. The post processing computations of the multiplier output is carried out at the ALU. The ALU unit is 32-bits wide and works on the output of multiplier and the other inputs. Since the proposed DSP supports basic arithmetic operations, the ALU unit is equipped with basic support configurations such as addition, subtraction and shifting. Two MUXs X_MUX and Z_MUX are used to select appropriate inputs for the ALU block.

The output of the ALU unit is fed to a MUX. This MUX provides the flexibility to select the output of the ALU or the multiplier output to drive the output port of the DSP block. The output side has two registers driven by a de-multiplexer (DEMUX) unit. Depending on the data occupancy of the registers, the DEMUX selects the register to hold the value. A load MUX is incorporated at the output side to select the values stored in these two registers. The selection of the MUX input lines depends upon the single bit select line value of *Load_MUX_selection*. This select bit provides the tractability to load the value stored in the register

from the previous cycle. Hence port E outputs the current result or previously calculated DSP result.

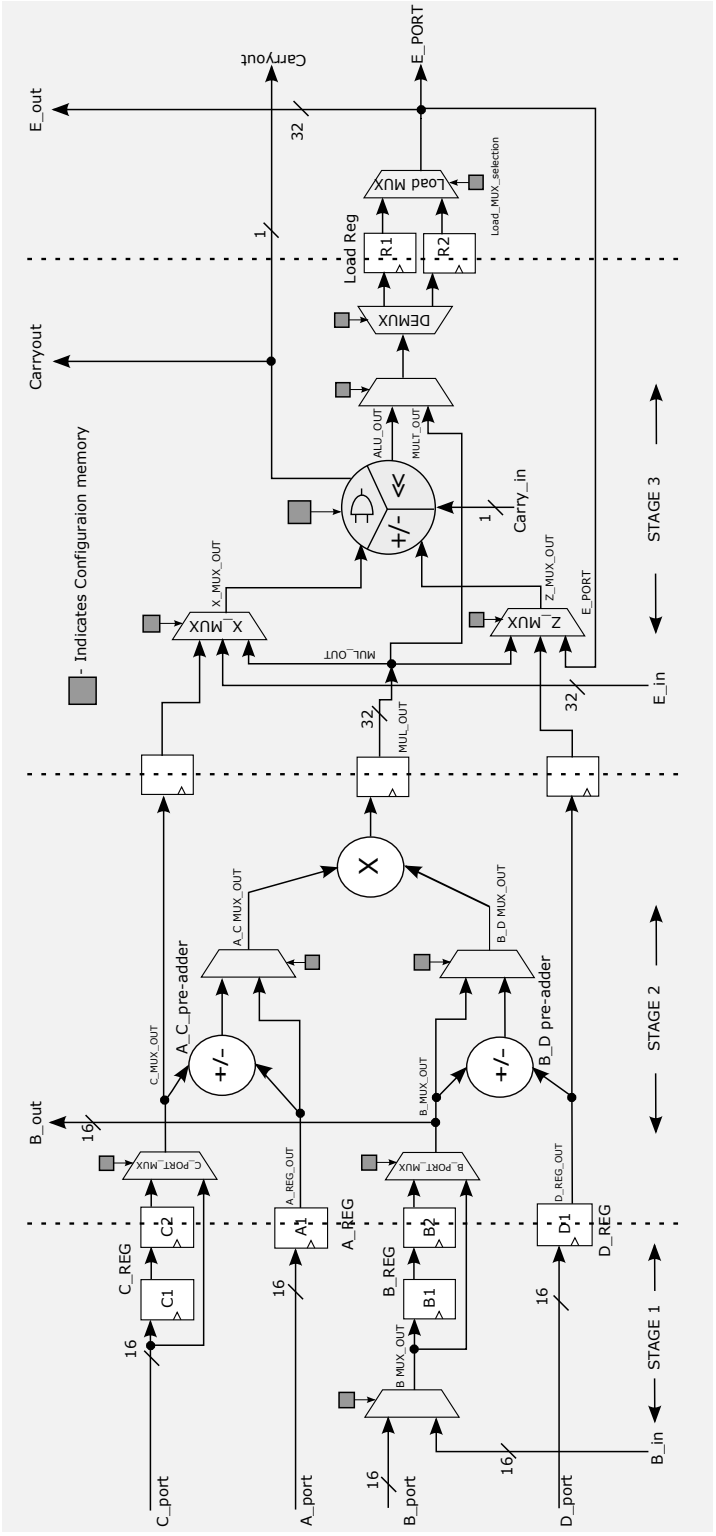


Figure 4.3: Detailed design of full-block DSP architecture.

Based on the input configuration bits, the sub-blocks (pre-adder, multiplier, post adder) are selected so as to realize the corresponding arithmetic operations. The configuration bits also contain the operation mode selectivity of the ALU which sets its functionality. Followed by the input stage pipeline, there exist one pipeline stage each at the output of the multiplier and the ALU unit. The DSP block operates with a fixed pipeline depth of 3 cycles for maximum operating speed, allowing it to be reused for an operation that is scheduled 3 or more cycles later.

(i) Input Ports

The main input ports of the proposed DSP block consist of the following: A_port, B_port, C_port, D_port, Carry_in, B_in, E_in, clock enable and reset inputs. Input ports A, B, C, and D are connected to the pre-adder unit through a register pipeline. The width of these input lines is 16-bits. B_in and E_in are cascaded inputs which are 16-bits and 32-bit wide respectively. These lines are useful when multiple DSP blocks are cascaded to implement complex operations like FIR filters. The carry_in port is a single bit wide. Ports A and C are tied to one pre-adder input while ports B and D are connected to other pre-adder. In case of the B port, signal passes through a MUX as it selects either cascaded input B_in or B port value to be passed to the pre-adder unit. The ports C and D are also input to ALU unit which can be selected by proper configuration of the MUXs X and Z. If the cascaded input is enabled, then E_in port is driven to ALU unit through X_MUX, while for MAC operation, the E port value is fed back to ALU through Z_MUX. Flexibility is provided at the multiplier output to bypass the ALU unit so as to get multiplication results directly at the output port E.

(ii) Output Ports

The main output ports of the DSP block are: E_port, carryout, B_out and E_out. E_port is 32-bit wide and serves as the main output of the DSP block. As mentioned in section 4.3.3, the E_port provides either the ALU result of the current

Table 4.1: Load MUX selection.

Load_MUX_selection Config[0]	E_Port
0	R1
1	R2

Table 4.2: DEMUX selection.

Config[1]	DEMUX_OUT
0	R1
1	R2

Table 4.3: Configuration control bits to select post ALU Multiplexer Output.

Config[2]	POST ALU MUX
0	ALU_OUT
1	MUL_OUT

Table 4.4: Operation modes of ALU.

Config[4:3]	ALU_OUT
00	X_MUX_OUT+ Z_MUX_OUT
01	(X_MUX_OUT + Carry_in)+ Z_MUX_OUT
10	(X_MUX_OUT + Carry_in)- Z_MUX_OUT
11	16 bit shift and add

cycle or the previous computation result stored in the output load registers. For cascading multiple DSP blocks, the output of one DSP is connected to the other through the output port E_out. Similarly, B_out is a cascaded output which is 16-bit wide, connected directly to the adjacent DSP blocks in case of computational intensive applications. The carryout pin is a single bit wide.

(iii) DSP Configuration bits

The proposed DSP block can perform 16-bit addition, subtraction and multiplication. Since the DSP is equipped with pre-adder units before the multiplier, the DSP can perform product of sum (POS) operations too. In many signal processing applications, the MAC operation is very common. By setting the configuration bits to select different components of the DSP, a MAC operation can be performed. The configuration bit setting for different units are mentioned in the Tables 4.1, 4.2 and 4.3.

Table 4.5: Configuration bits for X_MUX and Z_MUX.

Config[8:7]	**Z_MUX_OUT**	**Config[6:5]**	**X_MUX_OUT**
00	D_REG_OUT	00	C_MUX_OUT
01	E_PORT_OUT	01	E_in
10	MUL_OUT	10	MUL_OUT

Table 4.6: Configuration bits for pre-adder output selection.

Config[12:11]	**A_C Pre-adder MUX_OUT**	**Config[10:9]**	**B_D Pre-adder MUX_OUT**
00	A_REG_OUT	00	B_MUX_OUT
01	A+C	01	B+D
10	A_REG_OUT	10	B_MUX_OUT
11	A-C	11	B-D

Table 4.7: C and B port select bits.

Config[14]	**C_MUX_OUT**	**Config[13]**	**B_MUX_OUT**
0	C_PORT	0	B_MUX_OUT
1	C2	1	B2

The mode of the ALU unit is selected using 2-bit configuration which is included as a part of configuration bits. The different operation mode of the ALU is as shown in Table 4.4. The ALU unit performs the addition or subtraction of two 32-bit outputs from X_MUX and Z_MUX along with the carry_in bit depending upon the configuration bit setting. The X_MUX and Z_MUX are two 3-input 32-bit MUXs that provide the flexibility to choose different input line to be set to drive the input of ALU unit. The cascaded E_in signal is connected to one of the three input line of X_MUX where as for supporting MAC operation, the output port E_PORT is fed back to ALU via Z_MUX. The multiplier output is connected to both MUXs. The other signals connected to X_MUX and Z_MUX are C_PORT and D_PORT. The Table 4.5 shows a configuration bits setting for both the MUXs.

2-bits configuration is used for pre-adder and its multiplexer component input selection. The most significant bit (MSB) selects the operation of the pre-adder, decides whether to perform addition or subtraction. The least significant bit (LSB) configures the MUX that follows the pre-adder. Depending upon the selection of LSB, the MUX line switches either the pre-adder output i.e. A+C, A-C, B+D

Table 4.8: Configuration bits for B and Bin.

Config[15]	B_MUX_OUT
0	B_PORT
01	B_in

and B-D or A_REG_OUT or B_MUX_OUT output. The configuration bit values for pre-adder selection is shown in the Table 4.6 below.

The input ports A_PORT, B_PORT, B_in, C_PORT and D_PORT are passed to the pre-adder stage through input pipe line registers. The input port selection and its pipeline stage selection using configuration bits are shown in the Tables 4.7 and 4.8.

4.3.4 DSP Interconnect

The total delay of a circuit implemented on FPGA fabric is the sum of logic delay and routing delay. Among these, the circuit is mostly affected by the routing resource delay (routing delay). Hence to reduce the routing delay, careful design of the interconnect is very important. Based on the routing parameters and various routing resources, the architectures can be designed to be fast and area efficient. Some of the routing architecture features are:

- Type of wire segment used for routing.
- Number of wire segments in each channel.
- Connection box flexibility (F_c) and switch box flexibility (F_s)
- Type of routing switch : pass transistor, MUX, or tristate buffer.
- Size of transistor in the switches, metal width and spacing of routing wires.

Different types of wire segments: length-1, length-2, length-4, length-8 and long wires can be used for interconnecting CLBs and DSPs in a general FPGA architecture. The length of a wire segment is the number of logic blocks it spans.

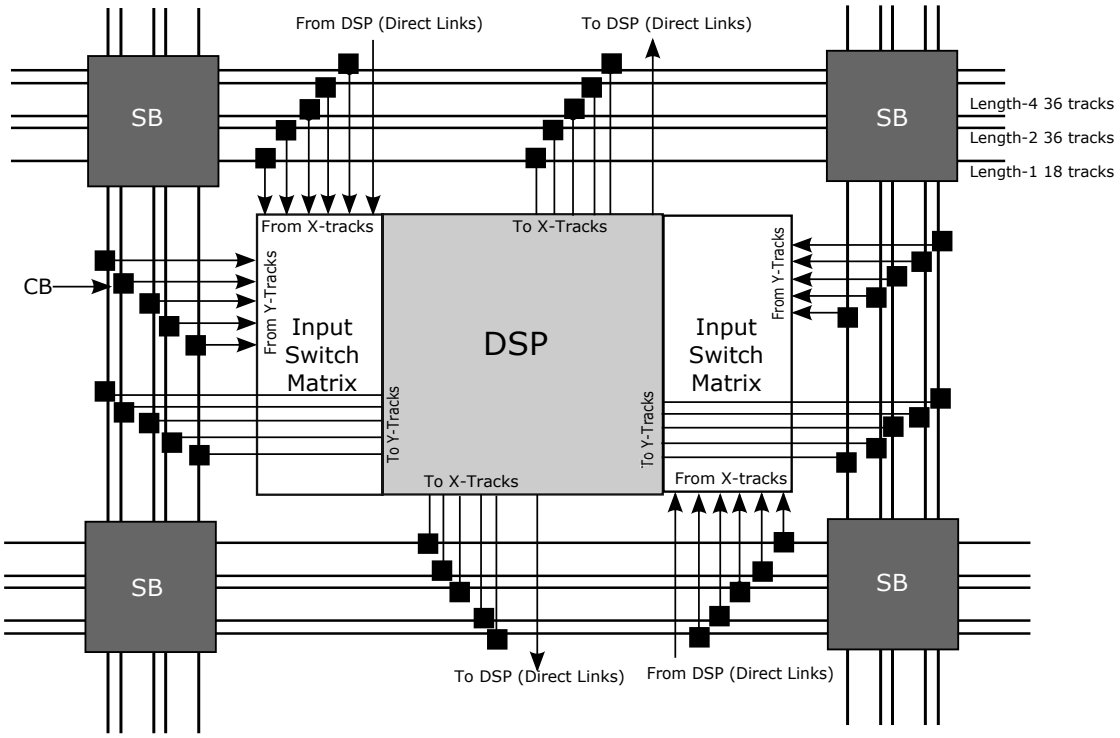


Figure 4.4: Interconnect design of DSP.

The performance of a circuit would be affected by using too many short wires as it requires a large number of routing switches for long transmissions. On the other hand, usage of long wires affects the routing flexibility and area of the circuit. Through design space exploration, a distribution of 20%, 40%, and 40%, for length-1, length-2, and length-4 wire segments was estimated to be the best for flattened NATURE architecture [9]. The interconnection between DSP and SMBs is routed using these wire segments, while interconnection between DSPs is routed internally (direct links).

While incorporating the DSP blocks in the flattened NATURE architecture, careful routing of wire segments is required to avoid routing congestion and delay. To address the number of wire segments in each channel (horizontal and vertical channel), it is required to determine the sufficient number of routing tracks required on one side of DSP block to avoid routing congestion. Based on the observations in Betz and Rose [179], Dehon and Rubin [180], the total number of routing tracks

on one side of DSP should satisfy the equation:

$$W_{min} = \frac{2 * Input + Ouput}{4} \tag{4.1}$$

The proposed DSP block has a total of 98 pins (input = 65, output = 33). Hence based on the equation 4.1, the total number of routing tracks on one side of the DSP $W_min = 40$. Based on the empirical estimation, additional tracks are added based to avoid routing congestion. [181]. Experiments performed on various benchmarks shows that a total of 90 vertical and horizontal tracks can satisfy the routing demand. As a result each adjacent channel of DSP contains 45 tracks which are divided in the ratio of 20%, 40%, and 40% between the length-1, length-2 and length-4 wire segments. The input or output pins of the DSP are connected to some selected wire segments or all the adjacent wire segments via a connection block (CB) [28] of programmable switches. There exists a switch block (SB) [28] at every intersection of horizontal and vertical channels. These are a set of programmable switches that allow some of the wire segments coming to the SB to be connected to others. By configuring appropriate switches, short wires segments can be connected together to form long connections. Signal flows from the logic block into the CB, and then into the routing channel and then to the SB which provides the change in direction. The connection box flexibility parameter F_c (defined as the ratio of tracks a pin of an SMB connects to and the number of available tracks for each type of wire in the adjacent channel) of the unified architecture is set to 0.25, so that each input/output pin of the DSP connects to all length-2 wires and length-4 wires. This allows good routing flexibility even though a large number of routing resource are required which in turn incurs area. The SB flexibility parameter F_s (defined as the number of wire segments to which each incoming wire segment can connect to) is set to 3. This implies that each segment entering the SB connects to three other segments. A simplified interconnect structure of this integration is shown in Figure 4.4. Apart from the general routing interconnects, SMBs are also connected to their 8 nearest neighbours using 24 direct links (horizontal, vertical and diagonal).

Hence, a successful routing of the proposed DSP block can be achieved by extending the routing architecture of the existing fine-grained NATURE platform.

4.3.5 Implementation Using Proposed DSP Block

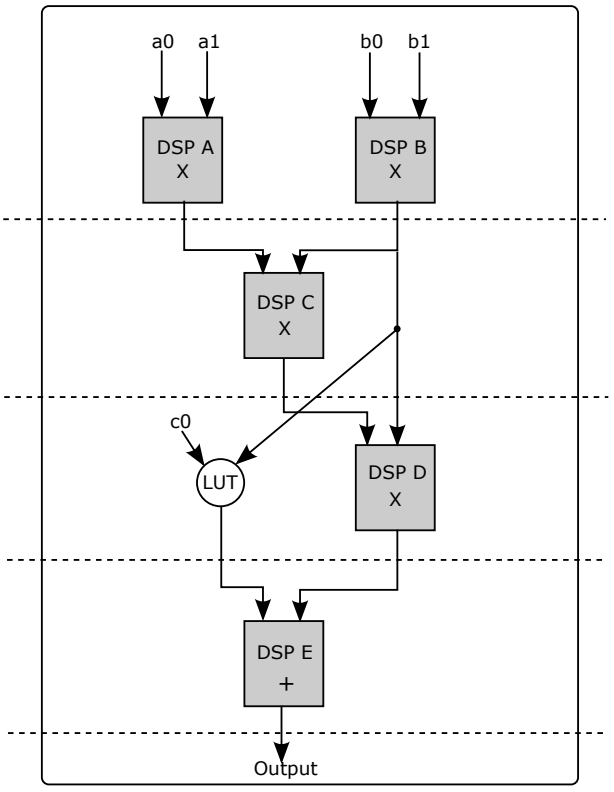


Figure 4.5: Illustration for DSP reconfiguration.

The design features of the proposed DSP architecture enhance the performance of compute intensive kernels when mapped onto DSP incorporating the NATURE architecture. The implementation of the design example in Figure 4.1 using the proposed DSP block is shown in Figure 4.5. The arithmetic operations (multipliers and adders) are mapped onto the proposed DSP blocks, while the multiplexer is mapped using LUTs. The delay of each DSP block is equivalent to 3 LUT computation cycles (due to 3 stage pipelining). Thus, the total delay of the implementation using DSP blocks is 12 LUT computation cycles compared to 25 LUT computation cycles when realized using LUTs only, reducing the delay by half.

One of the key features of the proposed DSP is its ability to reconfigure after every 3^{rd} clock cycles. The basic idea of DSP logic folding is that different non-overlapping arithmetic operations in the benchmark can be realized using same DSP block. In the scheduled data flow graph shown in Figure 4.5, mathematical operations in different stages are mapped onto DSP blocks forming a chain of DSP blocks. The DSPs A and B co-exist in same clock cycle whereas DSPs C, D and E perform operations in different clock cycles. Hence after the first cycle, using temporal logic folding, DSP A is reconfigured to perform operation of DSP C, D and E in the successive DSP operation cycles. Thus, using 2 full-block reconfigurable DSP blocks, the illustrated DFG in the Figure 4.5 can be realized. The direct mapping of the illustrated DFG would results in more number of DSP blocks (4 DSP blocks) on commercial FPGAs as the vendor tools are currently unable to analyze a generic compute kernel and automatically reuse the DSP blocks during the implementation phase. Since multiple functions can be realized by a single DSP through reconfiguration (stored in the NRAMs), instead of multiple DSPs as in traditional reconfigurable architectures, significant area saving is achieved.

4.3.6 Performance measurement

The total area including the switch matrix and the maximum operating frequency of the proposed DSP block is estimated using Synopsys Design Compiler simulation targeting the TSMC 65 nm technology library. Also, the dynamic power consumption of the DSP block is determined using the Synopsys PrimeTime tool. The switching activity file (vcd) is generated across a wide variety of input patterns, which are then provided to the PrimeTime tool to estimate power, using same technology library. Table 4.9 illustrates the total area (including input switch matrix), total dynamic power and maximum operating frequency of the proposed DSP block. The size of the DSP block is set to be the size of 6 SMBs in order to provide enough interconnections with surrounding tracks when integrated with fine-grained logic The total number of configuration bits for the proposed block

Table 4.9: Power, area and frequency of the proposed DSP block.

Total Cell Area (μm^2)	Total Dynamic Power (μW)	Max. Operating Frequency (MHz)
14209	1390	300

is 16-bits. Currently it is assumed that 16 copies of the reconfiguration bits are stored, and the configuration memory can be any high density, fast-access nano memories like NRAMs/PCMs/MRAMs. Typically, the inclusion of such nano RAMs incur less than 10% area overhead and reconfiguration delay is less than 5% of total delay [8].

4.4 DSP Applications

This section demonstrates how DSP blocks can efficiently support implementation of typical DSP functions.

4.4.1 32x32 bit multiplication

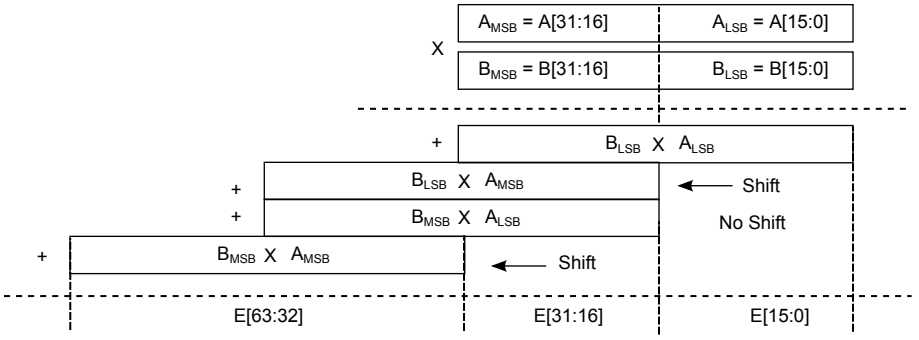


Figure 4.6: 32x32 multiplication using 16x16 multiplier.

Figure. 4.6 illustrates the implementation of 32x32 bit multiplication using 16x16-bit multipliers. With the aid of temporal folding, using only one DSP block, a 32×32 multiplication can be implemented as shown in the Figure 4.7. Since the input port of DSP is 16-bit wide, the 32-bit numbers to be multiplied are fed as 16-bit LSB and MSB respectively. After every DSP operation cycle (3- cycles), the reconfiguration is performed after the computed results are stored in the register

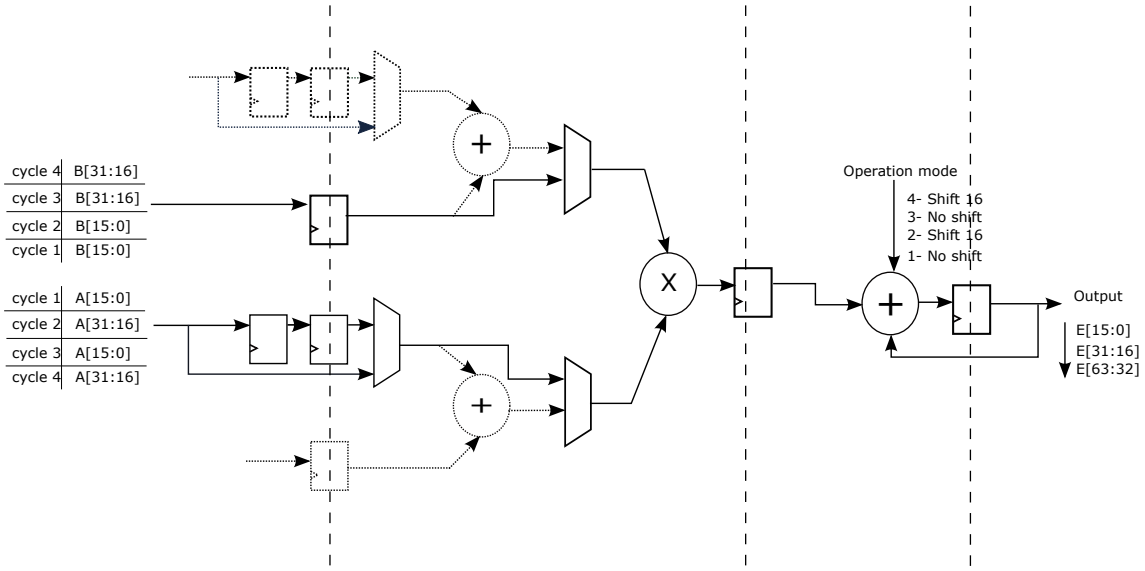


Figure 4.7: 32 bit multiplication using logic folding.

for later use. For each DSP life cycle, the operation of the ALU is changed from idle to shift mode for the corresponding bit addition to get the correct result. In the first clock cycle, the LSBs of both numbers are multiplied to get a 32 bit number. The 16-bit LSB of the output provides the lower 16-bits of the final result at the end of 3^{rd} clock cycle. In the second cycle, the MSB of A and LSB of B are multiplied, shifted and added with corresponding bits of previous cycle. MSB of B and LSB of A are multiplied and added with the accumulated result to generate the next 16-bits of the final result after ninth clock cycle. Bit shift is not performed between 6-9 cycles. The final DSP block perform multiplication of both MSBs, shifted and added with the previous result and fed to the output port to get the final 32-bit result. For the 32-bit multiplication, the proposed DSP slice takes 12 clock cycles to perform the operation. The timing diagram of 32 bit multiplication and its reconfiguration is illustrated in the Figure. 4.8.

DSP blocks in commercial FPGAs also support wider multiplication; however, a direct 32×32 operation in design is often mapped using multiple DSP blocks by the vendor tools. By leveraging dynamic programmability (on Xilinx DSP blocks), it would be possible to map the entire operation to a single DSP block at the cost of additional fine-grained resources [182].

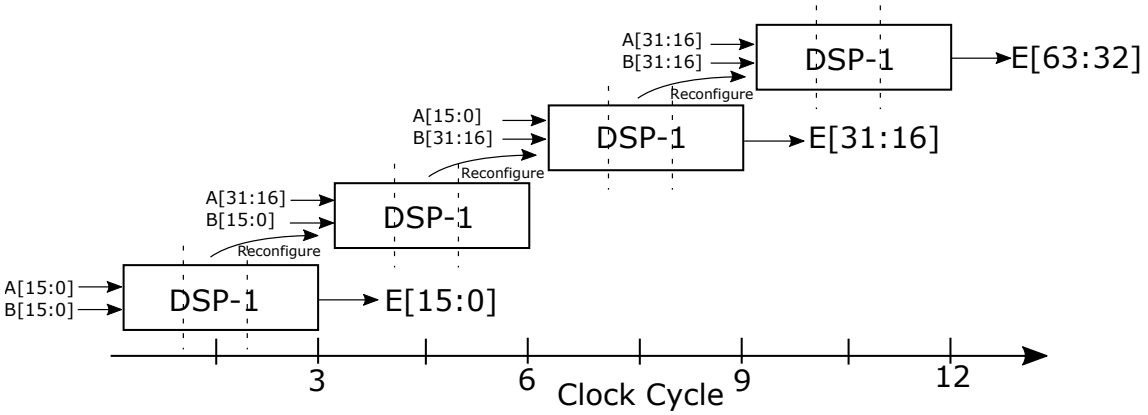


Figure 4.8: Timeline of the reconfiguration of 32 bit multiplication.

4.4.2 FIR filter implementation

The FIR filter has a finite length impulse response (or response to any finite length input is of finite duration), because it settles to zero in finite time [183]. The general FIR filter convolution equation is given by:

$$y(n) = \sum_{i=0}^{N-1} h(i)x(n-i) \tag{4.2}$$

The output at any time n is simply the weighted linear combination of the input signal samples *x(n), x(n-1),.... x(n-N+1).* The weight factor *h(i)* is the impulse response of the filter. Realizing equation 4.2 on a hardware platform requires a large number of arithmetic nodes (additions, subtraction and division) as it requires M DSP blocks and has a complexity of N multiplications and *N-1* additions per output point. However, there are a wide variety of FIR filter structures available. Some of them are cascade, parallel, lattice structures that are robust in finite word length implementations [184]. The type of architecture chosen is determined by the amount of processing required in the available clock cycles. One of the common filter structures used for high-performance applications is the fully parallel FIR filter [185]. The following section shows the implementation of transposed and symmetric systolic FIR filters using the proposed DSP block.

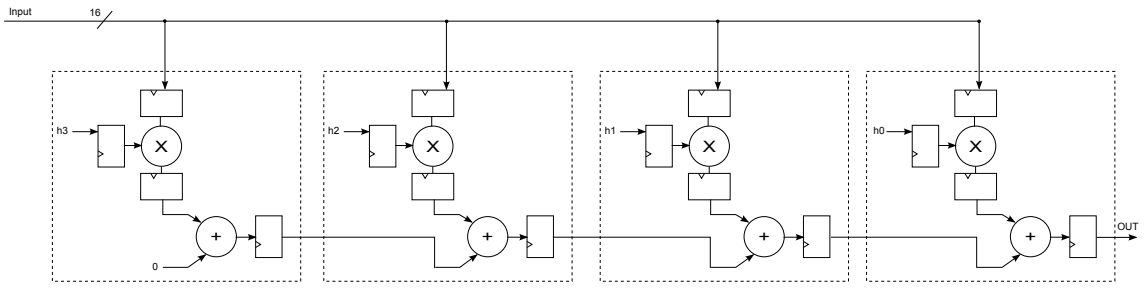


Figure 4.9: Transposed FIR filter.

(i) Transposed FIR filter

A variation of the direct FIR model is called the transposed FIR filter. It can be constructed from the direct form FIR filter by exchanging the input and output and inverting the direction of signal flow. The incorporation of registers between the adders of the transposed structure achieves high throughput without adding any extra pipeline registers.

The proposed DSP slice is designed in such a way that it can be efficiently chained together using dedicated routing resources. This makes it better to implement the transposed FIR structure. The transposed FIR implementation using chaining of the proposed DSP block is shown in Figure. 4.9. Here the 16-bit input is fed to each DSP slice simultaneously. It can be observed from the figure that the co-efficients are ordered from right to left with *h(0)* fed to the right most DSP and *h(N-1)* to the left most DSP. The result calculated from each DSP slice is fed to the pipelined adder chain which acts as a buffer to hold the inner product calculated during the previous cycle. Without inheriting any additional logic, the transposed FIR filter structure can be realized using DSP blocks.

(ii) Symmetric systolic FIR filter

Linear phase response is a desirable feature of FIR filters, where the phase response of the filter is a linear function of frequency [186]. Symmetric filters are very much favored as their phase response is linear, which is beneficial in many applications. An N order symmetric FIR filter expression is given by:

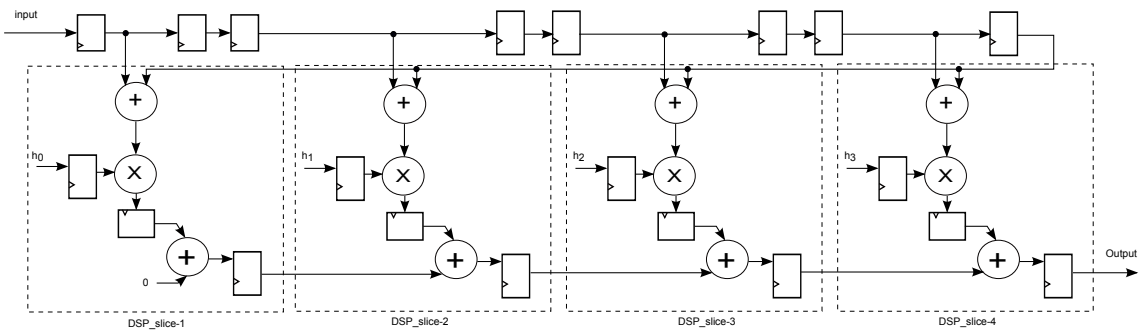


Figure 4.10: Symmetric systolic FIR implementation.

$$y(n) = \sum_{i=0}^{N-1} h(i)x(n-i) \tag{4.3}$$

whose unit impulse response satisfies the condition,

$$h(n) = h(N-1-n), n = 0, 1, 2, ..., N-1 \tag{4.4}$$

Different design methods and many alternative structures for symmetric filters are available in the literature [187, 188, 189, 190]. Making use of the symmetry is extremely powerful in parallel FIR filters as it reduces the required number of multipliers by half. Hence it reduces the number of DSP slices required to implement the symmetric FIR filters. Figure. 4.10 shows the implementation of the symmetric FIR filter structure using the proposed DSP slice. It can be observed from the structure that, before weighted linear combination, the input data is pre-added. The proposed DSP slice can efficiently implement the structure using one of the two pre-adder unit. Unlike the transposed FIR filter structure, the coefficients are arranged from left to right with *h(0)* fed to the left most DSP and *h(N-1)* to the right most DSP. Similar to the commercial DSP blocks, the incorporation of pre-adders in the proposed DSP structure allows the efficient mapping of symmetric FIR filter structure. In general, for a N (N being even) order symmetric FIR filter, the number of DSP slices required to realize it is (N/2)+1.

4.4.3 Illustrative Example

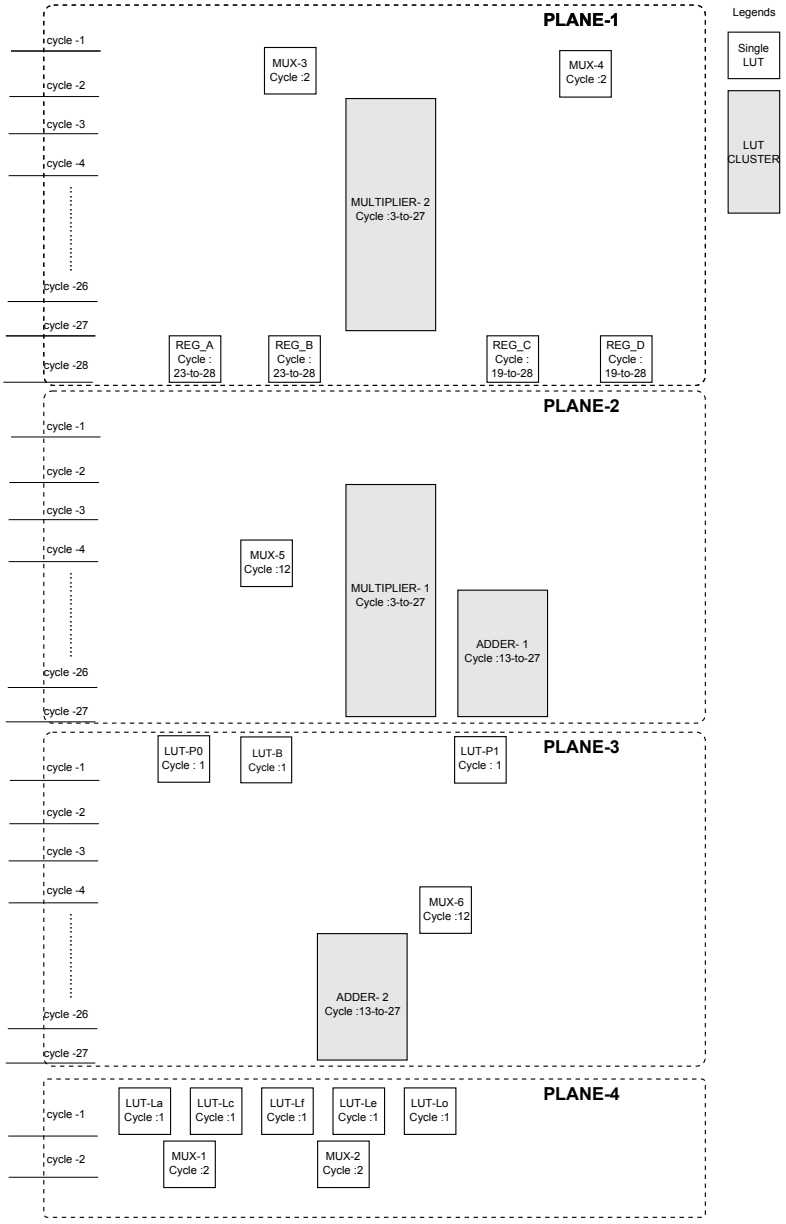


Figure 4.11: LUT/LUT cluster allocation of sample benchmark after scheduling.

In this section the performance of the NATURE architecture that incorporates the proposed DSP block is presented by mapping the result of a small benchmark circuit. The LUT/ LUT cluster folding cycle allocation in each plane after scheduling for level-1 folding is shown in the Figure 4.11.

In the existing fine-grained NATURE architecture, both the arithmetic and random logic are realized using fine-grained modules (LUT). Figure 4.11 illustrates

that the multiplier realization in folding level-1 using LUTs takes 25 folding cycles to get the final output (from cycle 3 till cycle 27). The other operations that follow the multiplier can only be scheduled once the result is obtained. Similarly for the addition node, it takes 15 clock cycles to be realized using LUT. It can be observed that the maximum folding cycle across all the planes is 28 in plane-1. Since the critical path lies along the arithmetic operations, the bottleneck that affects the delay while implementing the benchmark using a fine-grained NATURE architecture is the LUT implementation of arithmetic nodes (addition, subtraction, multiplication).

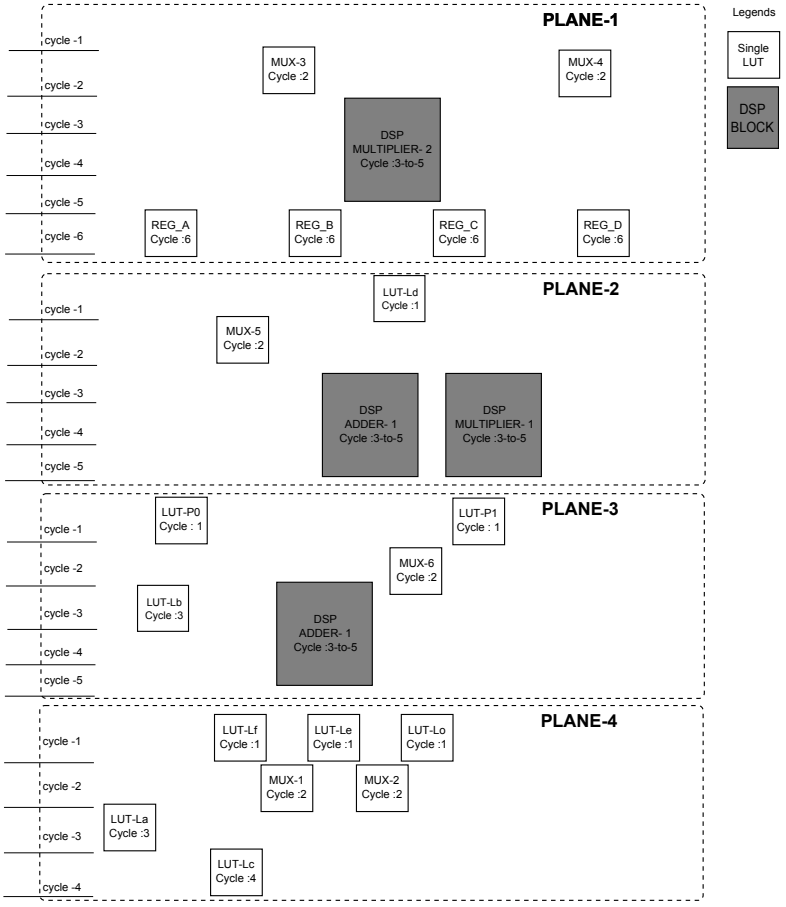


Figure 4.12: LUT and DSP allocation of sample benchmark after scheduling.

By incorporating the full-block reconfigurable DSP in the existing fine-grained NATURE architecture, all the mathematical operations can be mapped onto it efficiently. Since the proposed DSP takes only 3 LUT computations to perform, the folding cycle delay will also be reduced significantly, while the remaining random

logic will be realized using fine-grained structure as shown in the Figure 4.12. By mapping the complex operations on DSP, it can be observed that the largest folding cycle across the four planes after scheduling is reduced to 6 for the sample benchmark circuit.

4.5 NanoMap tool flow Enhancement for DSP Block

In the enhanced dynamic reconfigurable architecture containing both fine-grained and coarse-grained modules, the NanoMap needs to be modified to support the mapping of the arithmetic computations on DSP blocks. The complex mathematical computations (such as extended multiplications, POS calculations, MAC operations, FIR filter implementations etc.) in the circuit which can be efficiently calculated using DSP are identified and marked during the circuit search step. These entries are mapped onto DSP during the technology mapping procedure. The step wise modification of the NanoMap tool for the unified architecture is discussed in the following section.

Figure 4.13 presents the enhancements done to the NanoMap in order to efficiently map the arithmetic operations on the proposed full-block DSP. In logic mapping step, after reading the RTL circuit, the basic arithmetic functions are mapped onto DSP blocks if the input width greater than 7 bits. In case of folding level 0, mapping 8-bit (or lower) arithmetic operations onto fine-grain elements consume a maximum of 118 LUTs (30 SMB blocks). Realizing such smaller datapath (up to 8 bits) arithmetic operations using the proposed DSP block results in under utilization as more than half of the compute/data path within the DSP block remains unused. However, higher bit width ($\geq$ 8 bits) arithmetic operations like a 16 bit multiplier requires only 1 DSP block in place of 494 LUTs (124 SMBs), resulting in better performance and reduced area. Hence in the tool-flow, arithmetic operations with bit width greater than 8 bits are mapped onto DSP blocks, while smaller bit width operations are implemented using LUTs. Also, as the

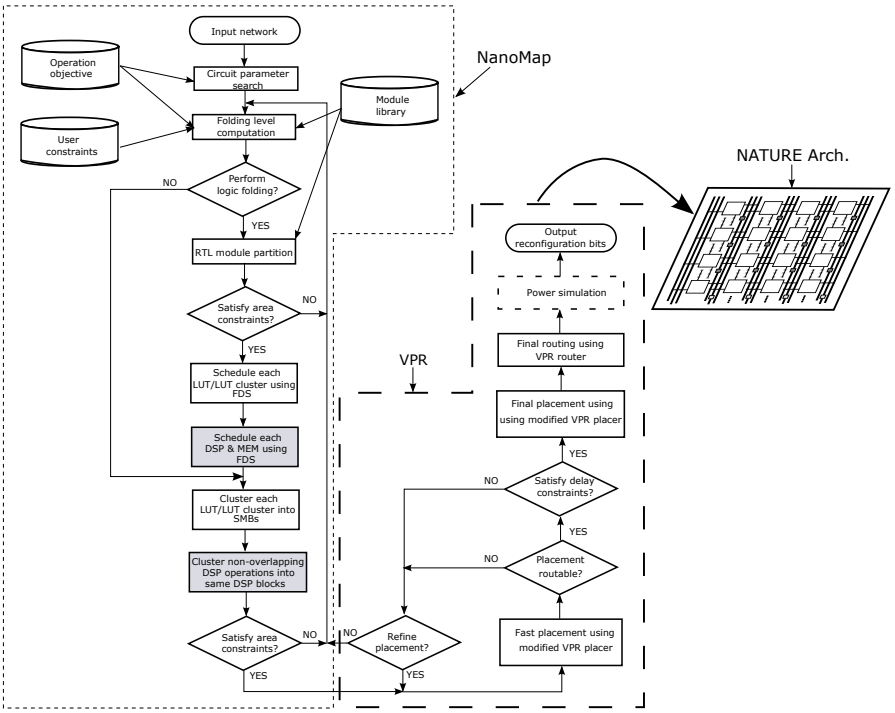


Figure 4.13: Mapping flow of NanoMap.

DSP slice has only one 16-bit multiplier and one 32-bit adder/subtracter, larger arithmetic operations need to be expanded based on the available functional units equipped in the DSP block or across multiple DSP blocks. In the logic mapping steps, the LUTs and DSPs are also scheduled into the best clock cycles to enable most resource sharing and reduce the total area usage using FDS [165]. The FDS is designed such that LUT and LUT clusters are given higher scheduling priority compared to DSP blocks. Because the number of LUT and LUT clusters used in the circuit mapping is more than the DSP blocks and has more constraints on its resource usage. The modified logic mapping and scheduling algorithm is shown in the Algorithm 1.

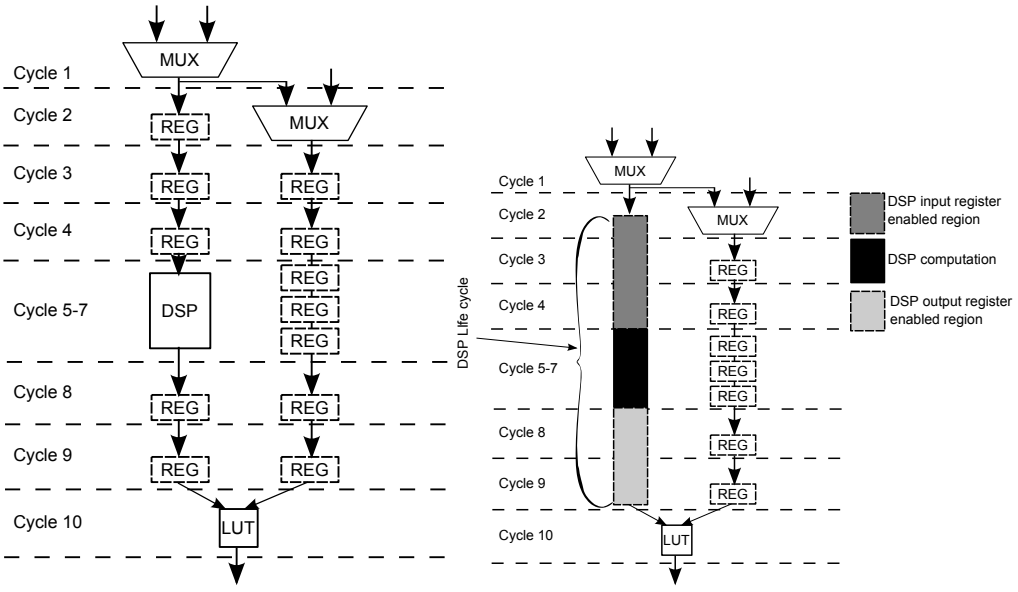
After scheduling, logic folding enables sharing of LEs and DSPs by different operations at different clock cycles. The clustering step that follows determines which DSP operations share each physical DSP block. In this step, the clustering algorithm considers the connections between DSP operations and the time span of DSP operation. In temporal clustering, the life cycle of DSPs is determined based on register storing and are grouped without causing a timing constraint. To minimize the LE resource usage, the DFF in LEs are not used for storing the values

```
while node is addition/subtraction/multiplication do
    if input width >= 8 then
        Mark DSP for the operation
    end
end
Expand the node
begin schedule  forall LUT/LUT cluster and DSP do
    Determine time frame of each using ASAP and ALAP scheduling
    Create LUT computation and register storage weights
    Create DSP computation weights
    foreach unscheduled LUT cluster, LUT and DSP computation i do
        foreach possible clock cycle j the nodes can be assigned to do
            calculate self-force of assigning node i to cycle j
            calculate predecessor and successor force
            Calculate total force for node i in cycle j
        end
        select the cycle with lowest force for node i
    end
end
```

Algorithm 1: DSP mapping and scheduling algorithm.

going to and from the DSP blocks . This is instead done by enabling the input and output registers of DSP to hold the value. The difference is illustrated in the two examples shown in the Figure 4.14a and Figure 4.14b. After scheduling of LUTs and DSPs, if any of the input to the DSP is arriving at a later stage than other inputs, the clustering algorithm will allocate the DSP blocks from its initial folding cycle to the earliest possible input signal stage so that the input registers within the block will hold the value rather than storing it in DFFs in LE. Similarly once the DSP performs the operation in its scheduled time frame and if its output is to be delivered to the LUT in a later cycle, the DSP block must remain active so that the DSP output registers hold the value till the corresponding folding cycle is reached.

The grouping of DSP blocks are performed after determining the life cycle of each node. When two DSPs are merged without any conflict, they are called *friendly*. All *friendly* DSPs are identified for each DSP. They are sorted in each folding stage in the ascending order of the number of *friendly* DSPs they have. If any DSP is not merged, the DSP with minimum grouping is chosen and merged into it.



(a) Without register enabled. (b) Register enabled.

Figure 4.14: DSP with and without input and output register usage.

This process of grouping continues till all the DSPs are considered for clustering. If there exist any timing conflict between DSPs, they are not considered *friendly* and hence not grouped together. The pseudo code for the algorithm is shown in the Algorithm 3. The output of temporal clustering is the netlist for placement and routing.

The final step in the mapping flow is the physical mapping in which the placement and routing is implemented on top of the VPR [119]. Modification of VPR has been done to enable mapping of circuit netlist containing DSP blocks. Blocks of different types, such as DSP blocks and LEs, cannot be swapped with each other during placement optimization. Only the same type of blocks can swap their locations. Furthermore, the layout of DSP is designed to support ease of connectivity between DSP and LE. The DSP blocks are arranged in vertical columns surrounded by LE blocks and routing structures. For routing, since the interconnection between DSP blocks are taken care of internally, the routing algorithm will consider only the interconnections between DSP and LEs. The routing delay estimation includes both DSP delay and intrinsic interconnect delays.

```
forall stages do
    foreach DSPs do
        while input/output edge coming/going from/to different cycle do
            Extend DSP life cycle to hold the value
        end
    end
end
forall stage do
    forall DSPs do
        identify its friendly DSPs
    end
    while there are unprocessed DSPs do
        pick DSP with less number of grouping
        if the DSP has a friendly DSP available then
            merge the DSP with the friendly DSP
        end
        mark the DSP as merged
    end
end
```

Algorithm 2: Temporal Clustering algorithm.

4.6 Experimental Results

This section demonstrates the performance effectiveness of the proposed full-block reconfigurable DSP incorporated NATURE (mixed-grained NATURE) architecture over the fine-grained NATURE platform on various benchmark suites. The experiment is performed on 13 benchmark suites, which contain large number of complex arithmetic operations. Among the 13 benchmarks, Diffeq and Paulin are differential-equation solvers [191], ASPP4 is an application-specific programmable processor [192], Biquad is a digital filter and wavelet is a mathematical functions that perform wavelet transform computations respectively. The 6 bigger benchmarks ARF, FIR1, FIR2, EWF, HAL, and Smooth Triangle are popular DSP applications obtained from UCLAs MediaBench which consists of a large number of addition, multiplication and subtraction nodes [193]. Out of the 13 benchmarks, GCD (greatest common divisor) is the smallest which contains only two comparators and one subtractor node.

Table 4.10: Comparison between mixed-grained and fine-grained NATURE architecture for FL-1.

Benchmark	Fine-Grained			Proposed Mixed-Grained				A×D	P×D
	SMB	Total Delay (ns)	Total Area (mm^2)	SMB	DSP	Total Delay (ns)	Total Area (mm^2)	Improvement	Improvement
ARF	126	44.75	315	0	28	23.25	397.6	1.52×	5.86×
DCT	100	48.62	250	2	32	32.48	459.4	0.81×	3.04×
EWF	73	52.50	182.5	0	34	29.04	482.8	0.68×	2.62×
FIR1	88	32.50	220	0	21	27.30	298.2	0.88×	3.37×
FIR2	68	45.25	170	0	23	27.36	326.6	0.86×	3.31×
Diffeq	29	72.40	72.5	13	3	57.40	75.10	1.18×	2.04×
GCD	8	16.38	20	8	1	78.12	34.2	0.12×	0.18×
HAL	48	36.00	120	1	8	86.45	116.1	0.50×	1.82×
Paulin	21	98.60	52.5	16	4	50.70	96.80	1.05×	1.86×
Wavelet	82	42.12	205	16	24	31.85	380.8	0.71×	2.11×
ASPP4	25	69.90	62.5	16	6	42.35	125.2	0.82×	1.65×
Biquad	19	54.04	47.5	8	4	37.76	76.80	0.89×	1.95×
Smooth Triangle	82	29.55	29.55	12	37	36.64	555.4	0.30×	1.01×

The area-delay product (A-D) and power-delay product (P-D) improvement of the proposed full-block reconfigurable DSP incorporated NATURE over fine-grained NATURE architecture for various folding levels (FL-1, FL-2 and FL-4) is explored. The results for FL-1, FL-2 and FL-4 are demonstrated in the Tables 4.10, 4.11, 4.12. The SMB and DSP block utilization, total delay (critical delay× total cycles), total area (in mm^2), A-D and P-D improvement of each benchmark suite for different folding levels (FL-1, FL-2 and FL-4) are illustrated in the tables. The total area (Total SMB × Area of one SMB + Total DSP × Area of one DSP) of the circuit is calculated after estimating the area of one SMB and one DSP block using Synopsys Design compiler on a 65 nm TSMC library as target technology. Similarly, to estimate the power consumption of different designs in the mixed-grained NATURE architecture, the power consumption of the DSP block and SMBs are computed using the Synopsys PrimeTime tool, as detailed in [194, 195]. The switching activity file (vcd) is generated across a wide variety of input patterns for both the SMB and the DSP block, which are then provided to the PrimeTime tool to estimate power using same technology node. By estimating the power consumption of these building blocks, the total power consumption of individual benchmark circuits on the unified NATURE is determined as $Power_{Total} = TotalSMB \times OneSMB_{Power} + TotalDSP \times OneDSP_{Power}$.

For FL-1, it can be observed from the Table 4.10 that realizing the benchmark using the fine-grained NATURE architecture results in significant reduction in area at the cost of total delay. This is because FL-1 results in more number of folding

Table 4.11: Comparison between mixed-grained and fine-grained NATURE architecture for FL-2.

Benchmark	Fine-Grained			Proposed Mixed-Grained				A×D	P×D
	SMB	Total Delay (ns)	Total Area (mm^2)	SMB	DSP	Total Delay (ns)	Total Area (mm^2)	Improvement	Improvement
ARF	252	27.43	630	0	28	14.13	397.6	3.08×	11.82×
DCT	200	28.86	500	3	32	14.34	461 .9	2.18×	3.08×
EWF	146	36.27	365	0	34	14.64	482.8	1.87×	7.19×
FIR1	175	24.44	437.5	0	21	13.77	298.2	2.60×	10.0×
FIR2	135	31.85	337.5	0	23	13.77	326.6	2.39×	9.18×
Diffeq	39	43.69	97.5	26	3	29.82	107.6	1.33×	1.88×
GCD	8	8.77	20	8	1	41.36	34.2	0.12×	0.18×
HAL	95	23.40	237.5	2	10	46.6	147	0.81×	2.84×
Paulin	42	52.56	105	36	4	24.3	146.8	1.55×	2.17×
Wavelet	172	23.40	430	35	24	14.01	428.3	1.68×	4.08×
ASPP4	50	37.95	125	34	7	19.9	184.4	1.29×	2.15×
Biquad	32	27.16	80	26	4	19.36	121.8	0.92×	1.41×
Smooth Triangle	139	16.32	347.5	12	37	18.52	555.4	0.55×	1.84×

levels allowing the SMBs to be scheduled with minimal overlap, resulting in better reuse of SMBs. In case of the mixed-grained NATURE architecture, though the DSP block enables reduction in delay, their area overhead and limited reconfiguration capability (once every 3 cycles) results in higher area consumption over the fine-grained implementations. This results in limited A-D improvement for the proposed mixed-grained NATURE at FL-1, especially for smaller benchmarks like GCD where the critical path is along fine-grained elements. However, the reduction in delay achieved by mapping computations to DSP block allows most designs to achieve improved energy efficiency (P-D) on the proposed mixed-grained NATURE architecture, as seen in Table 4.10. An average P-D improvement of 2.35× is achieved by the proposed mixed-grained over the fine-grained NATURE architecture.

Table 4.11 shows the comparison between mixed-grained and fine-grained NATURE architecture for FL-2. In case of FL-2, the number of folding cycles decreases causing more overlapping of SMB blocks, resulting in increased resource utilization of fine-grained elements. No significant change in DSP block utilization is observed in FL-2 over FL-1 as the reconfiguration of the proposed DSP block occurs only after every 3rd clock cycle. Thus, compared to fine-grained architecture, it can be observed that the A-D product for FL-2 has improved significantly over FL-1 for the mixed-grained NATURE architecture. The experiments show that the proposed mixed-grained architecture achieves an average A-D improvement of 1.56× and a maximum P-D improvement of up to 11.82× for FL-2 over

Table 4.12: Comparison between mixed-grained and fine-grained NATURE architecture for FL-4.

Benchmark	Fine-Grained			Proposed Mixed-Grained				A×D	P×D
	SMB	Total Delay (ns)	Total Area (mm^2)	SMB	DSP	Total Delay (ns)	Total Area (mm^2)	Improvement	Improvement
ARF	500	15.96	1250	0	28	9.60	397.6	5.23×	20.08×
DCT	400	22.54	1000	4	32	9.66	464.4	5.02×	18.19×
EWF	282	20.93	705	0	34	9.82	482.8	3.11×	11.95×
FIR1	348	20.93	870	0	21	9.18	298.2	6.65×	25.55
FIR2	268	21.91	670	0	23	9.54	326.6	4.71×	18.10×
Diffeq	54	26.55	135	44	3	13.68	152.6	1.72×	2.16×
GCD	8	9.12	20	8	1	22.98	34.2	0.23×	0.33×
HAL	188	18.20	470	3	10	25.05	149.5	2.28×	7.68×
Paulin	64	29.80	160	36	4	16.32	146.8	1.99×	2.79×
Wavelet	353	20.93	882.5	67	24	9.44	508.3	3.85×	7.64×
ASPP4	98	23.20	245	64	7	15.33	259.4	1.44×	1.99×
Biquad	64	18.32	160	26	4	10.16	121.8	2.37×	3.62×
Smooth Triangle	282	11.15	705	12	37	5.13	555.4	1.44×	4.80×

fine-grained NATURE architecture.

In case of FL-4, the number of folding cycles decreases further resulting in increased area and reduction in total delay, results of which are shown in Table 4.12. For example, realizing the benchmark ARF using FL-4 achieves 41.8% reduction in total delay with 1.98× increase in SMB utilization over FL-2 implementation for fine-grained architecture. However, the number of DSP block required to realize ARF benchmark across two folding levels (Fl-2 and FL-4) remains constant with 32% reduction in total delay. Across 13 benchmarks, an average A-D and P-D improvement of 3.08× and 9.6× is achieved for mixed-grained NATURE over fine-grained NATURE architecture for FL-4.

FL-0 configuration is similar to commercial FPGAs like Altera and Xilinx where cycle level reconfiguration is not performed. In case of the largest benchmark ARF, it is observed that FL-0 implementation consumes 2173 SMB blocks on fine-grained NATURE and only 28 DSP blocks on mixed-grained NATURE architecture. This translates to an A-D improvement of 10.92× and P-D improvement of 41.96× for mixed-grained NATURE over fine-grained NATURE architecture.

4.7 Summary

This chapter presented a full-block reconfigurable DSP block that can be incorporated with the NATURE architecture to efficiently map compute intensive

arithmetic operations over fine-grained NATURE architecture. In addition, the NanoMap tool is extended such that it is able to identify the complex mathematical functions and map them onto the proposed DSP block efficiently. The simulated results demonstrated that significant improvement in A-D and P-D product can be achieved for mixed-grained NATURE over fine-grained NATURE architecture. Compared with the fine-grained NATURE architecture, circuit realization for folding levels 1, 2, and 4 showed that incorporation of proposed DSP block can lead to respective improvements of 2.35×, 11.82× and 9.6× in P-D product. Similar improvements of 1.56× and 3.08× in A-D product is achieved for folding levels 2 and 4 respectively. In case of FL-1, the A-D product improvement is not significant in mixed-grained NATURE architecture due to the better overlapping of SMB blocks across different folding cycles in fine-grained NATURE architecture. A limitation of the proposed DSP architecture is its fixed 3-stage pipeline structure, which allows the DSP block to be reused only after 3 cycles, resulting in under utilization of the different stages.

5

Pipeline Reconfigurable DSP for NATURE

5.1 Introduction

As discussed in the previous chapter, the full-block reconfigurable DSP block takes advantage of the temporal logic folding of NATURE to enable the possibility to fold compute intensive arithmetic operations in time and map each fold to the same DSPs in the architecture. While this has increased the logic density by an order of magnitude, it is still short of fully exploiting the temporal logic folding.

For example, the 3 stage full-block reconfigurable DSP block described in chapter 4 produce result only at the end of 3^{rd} clock cycle, only after which the DSP block can be reconfigured to map other arithmetic operations. Hence full-block DSP block only allows mapping and folding of arithmetic operations that are not scheduled

within its operation cycles, resulting in under utilization of the DSP block, as only one pipeline stage within the design is active at any given clock cycle.

In this chapter, an enhanced reconfigurable DSP architecture for NATURE is presented which overcomes the limitations of the full-block reconfigurable DSP. The proposed DSP architecture allows independent reconfiguration of the individual pipeline stages to enable its full utilization. Thus, more operations can be folded onto the same DSP block, thereby minimizing the area overhead and increasing the effective logic density. Furthermore, the new DSP design also provides better support for complex multiplication as well as various filter designs. Compared to the previous full-block DSP design which uses a Wallace tree multiplier, the proposed DSP incorporates a Baugh-Wooley (BW) multiplier which offers better performance and area efficiency. Clock gating is also used at individual pipeline stages to minimize the power consumption. Finally, the NanoMap tool flow is also enhanced for mapping applications on the NATURE architecture by efficiently utilizing the proposed DSP blocks.

The work presented in this chapter is also discussed in,

1. R. Warrier, C. H. Vun, W. Zhang, *A Low-Power Pipelined MAC Architecture Using Baugh-Wooley Based Multiplier*, in Proceedings of the IEEE Global Conference on Consumer Electronics (GCCE), pp. 505-506, October 2014.

2. R. Warrier, W. Zhang, C. H. Vun, *Pipeline Reconfigurable DSP for Mixed-Grain Dynamically Reconfigurable Architectures*, Submitted to Springer Journal on Circuits, Systems and Signal Processing (CSSP) (Accepted).

5.2 Related Works

Most modern FPGAs are incorporated with high performance DSP blocks [5, 80] that efficiently implement complex arithmetic operations to enable acceleration for data-intensive computations. In [63], Ebeling *et al.* proposed a reconfigurable

pipelined datapath (RaPiDs) architecture which delivers very high performance for regular compute intensive applications. Word based computations of RaPiD consists of coarse-grained functional units that are arranged linearly over a programmable interconnects and datapaths. Intermediate results are stored locally in registers and small RAMs, located in their destination functional units. However, due to the absence of fine-grained units, RaPiD is not able to map highly irregular computation kernels. Another coarse-grained reconfigurable architecture to achieve extreme performance speedup for various compute intensive application is PipeRench [196]. PipeRench architecture incorporates the technique of pipeline reconfiguration, which involves virtualization of a large logical configuration on a small piece of hardware through rapid configuration. The core architecture consists a number of strips, with each strip constituting a group of processing element (PE) and programmable interconnects. During each pipeline reconfiguration, the strips are reconfigured to perform scheduled computations, with intermediate registers to hold the values of previous computation. However, the major bottleneck that limits the widespread adoption of PipeRench is the limited bandwidth between the architecture and main memory. Commercial vendors like Altera and Xilinx provide high performance DSP blocks [197, 5], with Xilinx supporting dynamic programmability in its DSP48 blocks. However, the Xilinx tool flow do not automatically support cycle-level reconfiguration, which has to be manually performed by the designer through the configuration input pins. This limits the scope of dynamic reconfigurability within a high-level design, reducing the area and power improvements that could be achieved in a design that incorporates large number of overlapping (or non-overlapping) arithmetic operations.

With the emerging technology trend for which power dissipation is a major concern, many techniques are proposed to achieve low power consumption in digital systems at a device level to system-level [198, 199, 200]. Power reduction techniques can be employed at different levels of DSP unit. In [199] low power consumption is achieved by using a controller that efficiently bypasses the multiplier and accumulator unit depending upon the consecutive input bits. Spurious Power Suppression Technique (SPST) using AND gate detection was employed in [201].

Hoang [202] proposed a low power MAC architecture using a carry save adder that replaces conventional carry propagation adder schemes. However these architectures occupy a large area as they incorporates logic for power management.

In this chapter, an enhanced DSP architecture that supports pipeline reconfiguration at runtime is presented. Efficient power management schemes to reduce the power dissipation of unused pipeline stage of DSP block are also discussed.

5.3 Pipelined DSP for NATURE

This section first presents the need for pipeline reconfigurability and power optimization for DSP block that support fast dynamic reconfiguration. Further sections explain the design features of proposed DSP architecture and its performance measurement. The pipeline reconfigurability and power optimization achieved after architecture enhancements and its interconnect design details are also presented. Finally the enhanced functionality of the proposed DSP block for implementing complex number multiplication is also discussed.

5.3.1 Motivational Example

The main feature of the NATURE architecture is its support for cycle level reconfiguration. The reconfigurability of LUTs at cycle level allows the designer to achieve area-delay trade-offs. However, the full-block reconfigurable DSP architecture incorporated in NATURE does not support such cycle level reconfiguration. These DSP blocks can be dynamically reconfigured to map other operations only after its full DSP cycle (3 clock cycles), as the individual pipeline stages cannot be enabled/disabled through the configuration bits on a per cycle basis. This results in under utilization of its pipeline stages. This can be explained with an example illustrated in Figure 5.1. Consider the Data Flow Graph (DFG) in Figure 5.1a to be mapped on the DSP incorporated NATURE architecture. Here each node denotes a mathematical or a logical operation. As shown in Figure 5.1b, the circuit

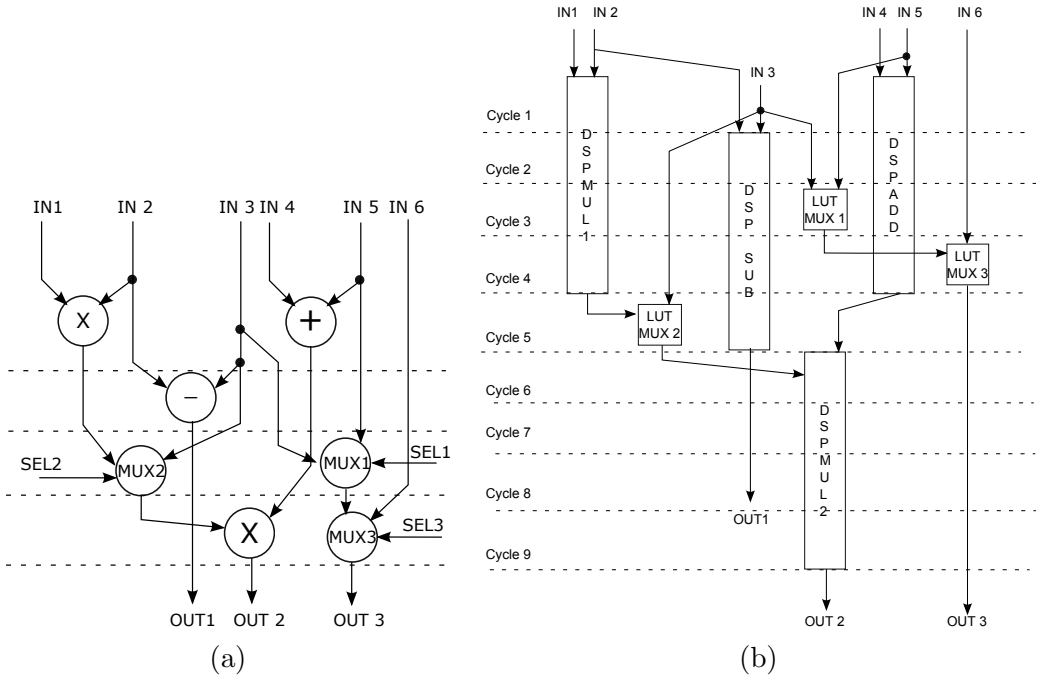


Figure 5.1: (a) DFG. (b) Scheduled DSP and LUT network.

is scheduled and mapped using LUT and DSP block for folding level-1. During logic mapping stage, the arithmetic nodes (addition, subtraction, and multiplication) are mapped to DSPs while other logical operations are mapped to LUTs. Hence, the arithmetic operations, DSP_MUL1, and DSP_ADD are scheduled to start the computation in cycle 1, DSP_SUB, and DSP_MUL2 scheduled in cycle 2, and 6 respectively. Similarly, the multiplexers (MUX1, MUX3, and MUX2) implemented using LUTs are scheduled in cycle 3, 4, and 5.

The temporal clustering stage that follows logic mapping determines LUT/DSP operations that share the same physical LUT/DSP block. Due to the overlapping life span of DSP_MUL1, DSP_ADD, and DSP_SUB operations, they are mapped to 3 separate physical full-block DSP blocks. However, the life cycle of DSP_MUL2 does not overlap with any of the other DSPs, and hence, it can be clustered with any other DSP block. As a result, it takes three DSP blocks to realize the DSP operations using full-block reconfigurable DSPs. As shown in the Figure 5.2, DSP-1 initially maps the operation DSP_MUL1 scheduled in cycle 1 and is reconfigured in cycle 6 to DSP_MUL2. However, other DSP operations scheduled in cycle 1 and

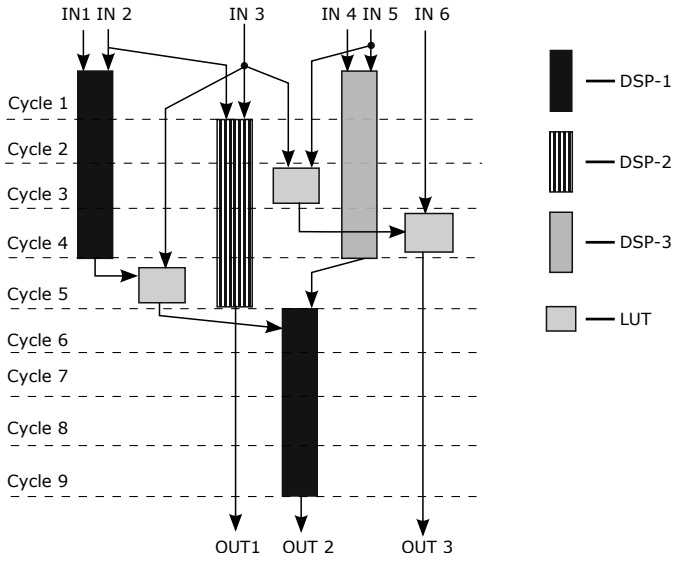


Figure 5.2: Implementation using full-block reconfigurable DSP.

2 (ADD and SUB) are independently mapped to two DSP block, DSP-2 and DSP-3 respectively. Hence, mapping this circuit using full-block reconfigurable DSP results in utilizing more DSP blocks since it supports reconfiguration only after full DSP operation. This chapter introduces a pipeline level DSP architecture that allows the reconfiguration of independent pipeline stages. This extension allows the mapping of DSP operations that are at least 1 pipeline stage apart, enabling optimal use of different DSP stages. Hence, more operations can be scheduled on to the same DSP block to achieve better area efficiency. Although the DSP48E1 blocks on Xilinx FPGAs support dynamic programmability, user designs have to manually exploit them using additional logic to drive the configuration pins on a per-cycle basis. It has been shown that vendor tools are unable to exploit the capabilities of the DSP block from a generic RTL description [203], achieving efficient mapping only in very specific cases (where the RTL code explicitly models the pipelined structure of the DSP block). Also, the vendor tools are currently unable to analyze a generic compute kernel and automatically reuse the DSP blocks during the implementation phase.

For any FPGA platform, design of a DSP architecture with high performance and low power consumption is a major challenge. Design tools map compute intensive circuits with performance constraints to DSP blocks rather than to other fine-grained elements, resulting in high DSP utilization. For such DSP intensive

implementations, the major power consumption is contributed by the DSP blocks. Typically, all pipeline stages of DSP blocks may not be used in all cycles when operations are mapped onto DSP blocks. However, this does not reduce the power consumption of the DSP block as the clock signals of these unused pipeline stages remain active and thus, power consumed by the unused pipeline stages of the DSP block also adds to the total power consumption of the circuit.

As an example, consider the case shown in the Figure 5.2. The DSP1 is scheduled across cycle 1 and 4 with 4 pipeline stages in each cycle. After the inputs IN1 and IN2 are processed in cycle 1, the pipeline stage 1 of DSP1 is not used until it is reconfigured in cycle 6. Similarly, for DSP2 and DSP3, the pipeline stages are unused till one DSP operation is completed. Since there is no active power management technique in the individual pipeline stages of the DSP, the power consumption of the DSP block fairly remains the same in all these cycles. Thus an active power management mechanism must be incorporated at each stage to minimize the power overhead of unused stages.

The pipeline reconfigurable DSP architecture proposed in this chapter allows the individual pipeline stages to be clock-gated to minimize the power consumption when the stages are idle. The control signal for individual stages is also integrated into the configuration bits of the DSP block. The detailed architecture of the pipeline reconfigurable DSP block is discussed in the next section.

5.3.2 Proposed DSP Architecture

Figure 5.3 shows the basic block diagram of the proposed DSP block. Similar to the previous DSP block, the proposed DSP block has 4 input ports and an output port. To support the pipeline reconfiguration, load-store registers are incorporated in the design. As shown in the Figure 5.4, the load-store registers use a multiplexer to select between the input data path and the values stored in DFF. This allows the flexibility to hold the value for later computational stage while the current pipeline stage is reconfigured. In the proposed DSP block, all the

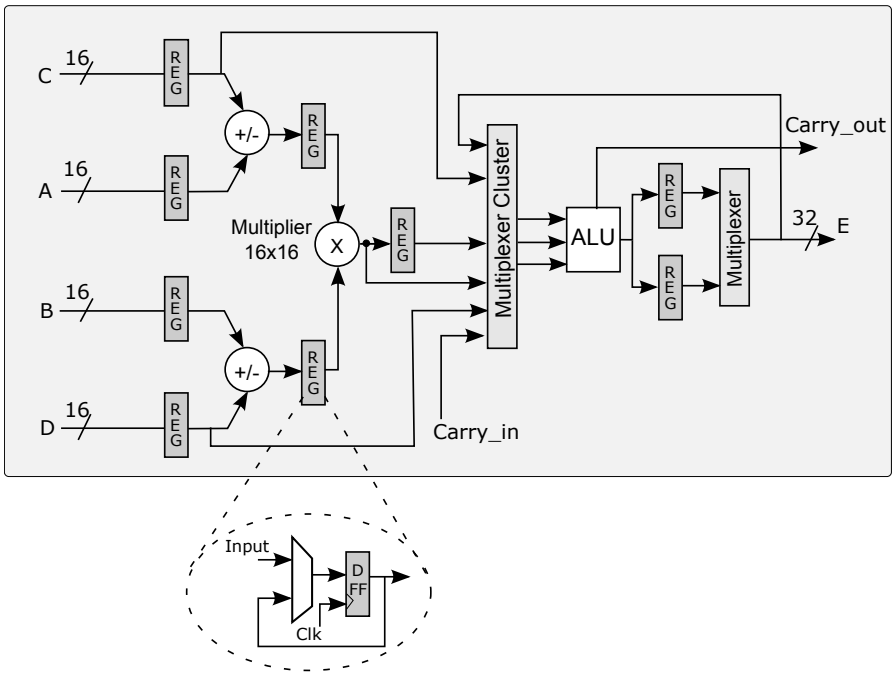


Figure 5.3: Basic structure of pipelined DSP block.

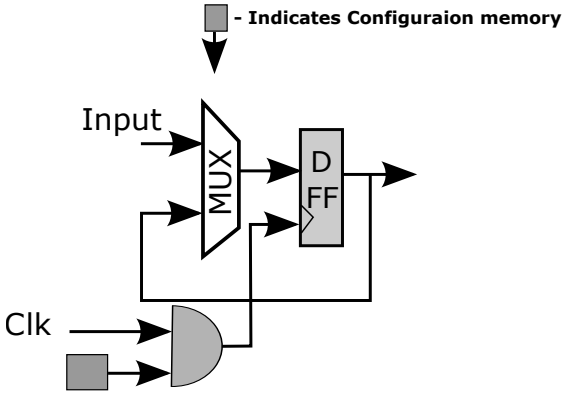


Figure 5.4: Load-store register design.

pipeline registers (DFFs) are replaced with load-store registers to enable pipeline level reconfigurability. Unlike the case with Xilinx DSP blocks where the pipeline registers are fixed at design time (through the bitstream), the individual pipeline stages can be enabled/disabled through the configuration bits on a per cycle basis. The proposed DSP block which is configured with all pipeline stages enabled (4-stage) at a particular cycle can be reconfigured at a later clock cycle in 1-, 2- or 3-stage pipeline mode. This allows the proposed DSP block to be flexibly reused across different cycles by scheduling the operations even with overlapping computations.

In order to minimize the power consumption of the DSP block, the unused pipeline stages are clock gated. Each load-store register is gated with a clock signal using

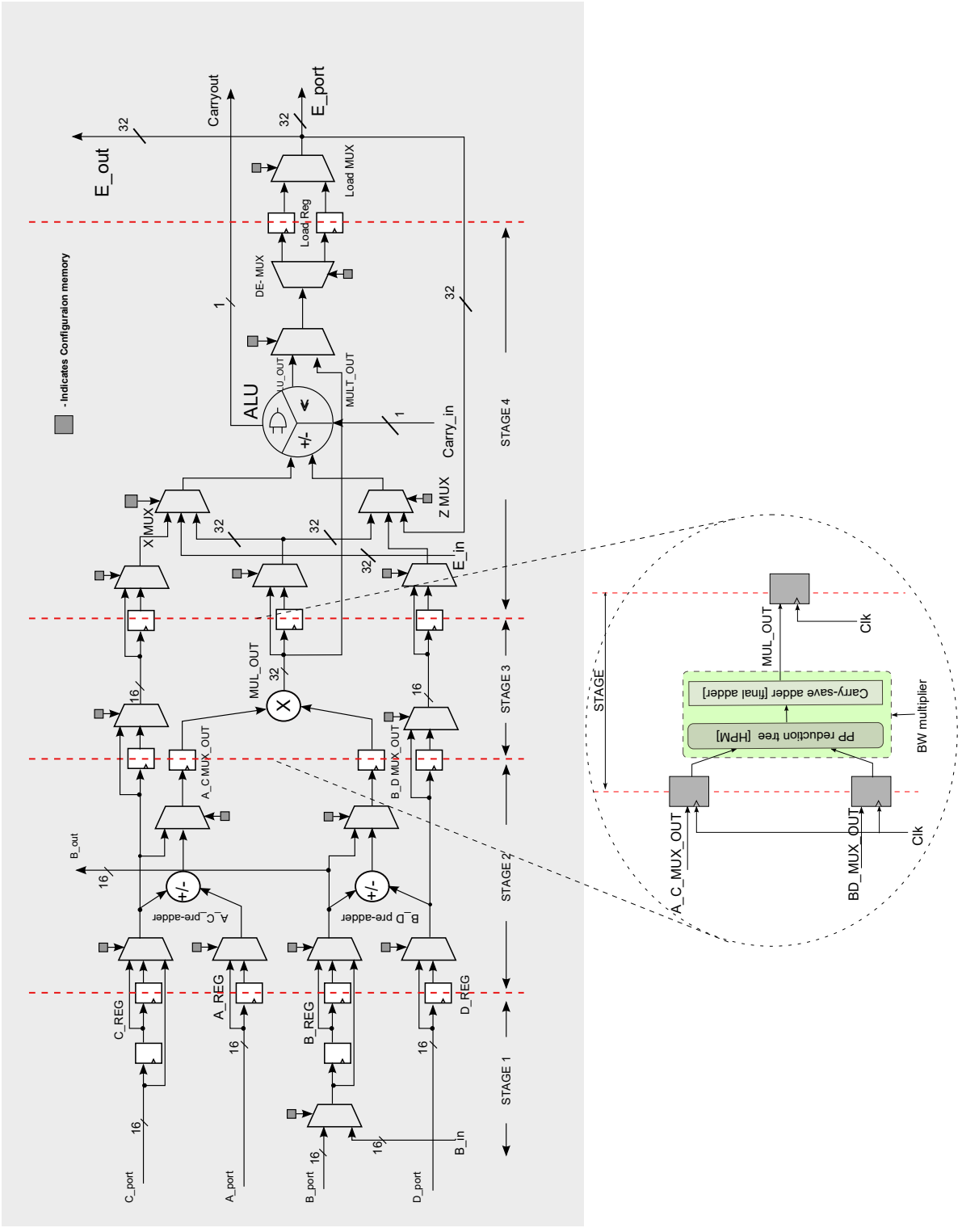


Figure 5.5: Detailed architecture of pipelined DSP block.

the corresponding configuration bits as shown in Figure 5.4. The clock gating signal is generated by ANDing the main clock signal (Clk) and the configuration bit. Hence, by using appropriate configuration bits, the inactive pipeline stages can be switched off thereby reducing the power consumption of the proposed DSP block.

The detailed architecture of the proposed DSP block is shown in Figure 5.5 and the individual units are explained below.

- 65 primary input pins include 16-bit *A_port*, *B_port*, *C_port*, *D_port*, and 1-bit *Carry_in*.
- 33 primary output pins comprise one 32-bit *E_port* and a 1-bit *Carry_out*.
- Two cascade input ports (16-bit *B_in* and 32-bit *E_in*) and output ports (16-bit *B_out* and 32-bit *E_out*) for chaining operations.
- Two 16-bit pre-adders/subtractors that support complex arithmetic operations. Four 16-bit input ports provide the input to the two pre-adder units.
- 32-bit ALU unit equipped to perform various functionalities such as addition, subtraction and 16-bit barrel shifter. It also supports bit-wise operations such as AND, OR, NOT, XOR, XNOR and other complex bit-level operations. The operation mode determines the functionality of ALU unit.

Compared to the previous design which uses a Wallace tree multiplier, the pipelined DSP block is incorporated with a BW multiplier with HPM reduction tree [173]. It supports both signed and unsigned multiplication and exhibits better performance than a Wallace tree multiplier and less power dissipation and smaller area than a modified Booth multiplier [204].

The major bottleneck that affects the performance of a DSP block is the multiplier unit. The performance of the previous DSP design which has 3 pipeline stages is affected by the critical path that lies between the pre-adders and the multiplier. With the design focus on improving the performance (in terms of speed) of the

Table 5.1: Power, area and frequency of the proposed DSP.

Total Cell Area (μm^2)	Total Dynamic Power (μW)	Max. Operating Frequency (MHz)
16487.2	1160	333

proposed pipeline DSP architecture, additional pipeline registers (load-store registers) are incorporated between pre-adders and multiplier unit. This enhances the DSP block to a 4 pipeline stage architecture achieving better performance compared to the previous design.

To provide interconnect flexibility, a number of multiplexers are incorporated in the DSP architecture for directing data to various stages. Unlike the conventional DSP designs, internal interconnects of the proposed DSP block provide more flexibility for efficient utilization of different functional units of DSP. Besides, some dedicated internal interconnects are used to bypass the multiplier block and to further optimize the power consumption. For example, for a combination of consecutive add/sub operations, instead of multiplying by *1* [172] or incorporating special control logic to bypass the multiplier [199], multiplier unit can be bypassed using dedicated interconnect lines, thus saving power.

5.3.3 Performance Measurement

The total area including input switch matrix and maximum operating frequency of the proposed DSP architecture is determined using Synopsys design compiler on a 65 nm TSMC library. Also, the total dynamic switching power is estimated using vcd dump file on prime time complier for the entire DSP design. Table 5.1 illustrates the total area, dynamic power and maximum operating frequency of the proposed DSP block. The proposed DSP design incurs an area overhead of 16.2%, as it incorporated a BW multiplier with HPM reduction tree which require additional circuitry over the Wallace tree multiplier based full-block DSP design. Moreover, with 16 reconfiguration copies, NRAM incurs less than 10% of area overhead and the reconfiguration delay is less than 5% of the total delay.

5.3.4 Pipelined Reconfiguration

A modified approach to partial reconfiguration (PR) of FPGA design is pipeline reconfiguration [196, 205, 206], in which the reconfiguration occurs in increments of pipeline stages and each pipeline stage is reconfigured as a whole. Pipeline reconfigurable computing is mainly used in a datapath-style computation where the number of pipeline stages required is larger than the available hardware resources. The architectural enhancement of the proposed DSP design incorporates pipeline reconfiguration allowing the 4 pipeline stages to be reconfigured independently. The pipeline reconfigurability of the proposed DSP block is designed to mimic the capabilities of the NATURE architecture, which supports reconfiguration of fine-grained elements in every clock cycle or in every few clock cycles, which cannot be supported by the full-block reconfigurable DSP block (presented in Chapter 4) due to its fixed pipeline structure. Unlike the full-block DSP block pipeline registers, the proposed DSP block has load store registers in each pipeline stage to hold the values that can be used for later stages. Of the four pipeline stages of DSP, the first stage is configured during the first clock cycle after which it begins computation. Once the execution of the first pipeline stage is finished, data flows to the second stage for further computation in the successive clock cycle. In this cycle, the first stage can be reconfigured to alter the processing flow for the next incoming data. This process can be repeated for all the pipeline stages of the DSP. When the pipeline stages are reconfigured, the interconnects can also be reconfigured by using appropriate multiplexer selection bits, enabling more flexibility in choosing the operation and/or data flow during each stage. The load-store registers incorporated in the pipeline stages of the proposed DSP allow the reconfiguration of individual pipeline stages without losing intermediate data, thereby lowering the resource utilization. The configuration bits for the DSP blocks are loaded from the NRAM on a per-cycle basis, similar to the fine-grain elements in NATURE architecture, resulting in a reconfiguration operation at each clock cycle. In conventional DSP architectures, the designers would require building external logic around the computational path to enable reuse of the individual stages (through

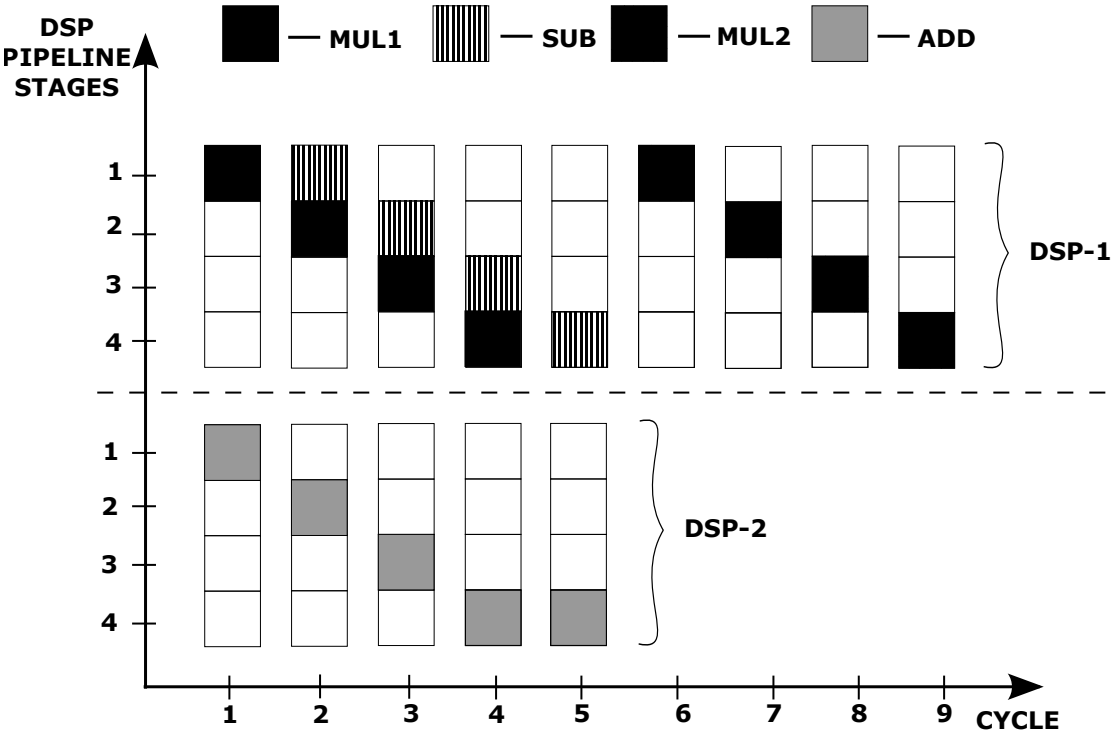


Figure 5.6: Timeline of the reconfiguration of DSP pipeline stages.

dynamic programming, where the configuration bits are driven by user-logic), thus increasing overall complexity and resource utilization.

Based on the previous design example illustrated in Figure 5.1, the pipeline reconfiguration capability of the proposed DSP gives the freedom to cluster more DSPs. In this case, all other DSP operations except DSP_MUL1 and DSP_ADD blocks can be clustered. Since the life spans of DSP_SUB and DSP_MUL1 differ by a pipeline stage, pipeline reconfiguration allows the arithmetic operations to be mapped to the same DSP block by reconfiguring the first stage. As shown in Figure 5.6, after clock cycle-1, the DSP_MUL1 stage-1 is reconfigured for SUB operation in cycle-2, while the successive stages are configured to realize MUL1 node. After the IN1 and IN2 signals moves to pipeline stage 3 (which contain multiplier block) in clock cycle 3, the stage 2 will be reconfigured to perform add operation of IN2 and IN3 using pre-adder unit in stage-2. The load store registers in stage-1 hold the intermediate values for successive stage processing. At clock cycle-6, the first pipeline stage of DSP-1 is reconfigured for MUL2 operation. Similarly, the ADD operation scheduled in clock cycle-1 will be mapped to DSP-2 block. As

the folding stage progresses, the reconfiguration of the successive pipeline stages allows sharing of the DSP block without data violations.

Hence compared to full-block reconfigurable DSP block, the proposed pipeline reconfigurable DSP block can realize the circuit with fewer DSP blocks, thereby reducing the total area and power. However, each pipelined DSP block incurs an area overhead of 6.7%, as it incorporates load-store registers which require additional multiplexers to support pipeline reconfiguration. Furthermore, the pipelined DSP block is register balanced by incorporating registers at the output of pre-adders to reduce the critical path delay and improve the performance. This also results in a slight increment in its area when compared to a full-block DSP block.

5.3.5 DSP Power Reduction Technique

In any FPGA platform, design of a DSP architecture with high performance and low power consumption is a major challenge. In the literature, power reduction techniques have been employed at different levels for DSP architectures. Since Multiply-and-Accumulate unit (MAC) is the most power consuming unit, many researchers aim to improve energy efficiency of the MAC units using various architecture-level enhancements. In [199], the authors proposed reducing the power consumption by bypassing the multiplier and accumulator depending on consecutive input bits. Clock gating method was introduced in [207] to minimize the power utilization of a multiplier in a MAC unit without using additional circuitry. The proposed DSP architecture also extends this approach by allowing each pipeline stage to be independently clock gated, thus providing finer control over power consumption.

The gate control is generated based on the configuration bits of the individual stages, allowing any unused pipeline stage(s) to be independently disabled. This approach, combined with the selectable routing within the DSP block allows data to be re-routed across a disabled stage. This is contradictory to conventional DSP

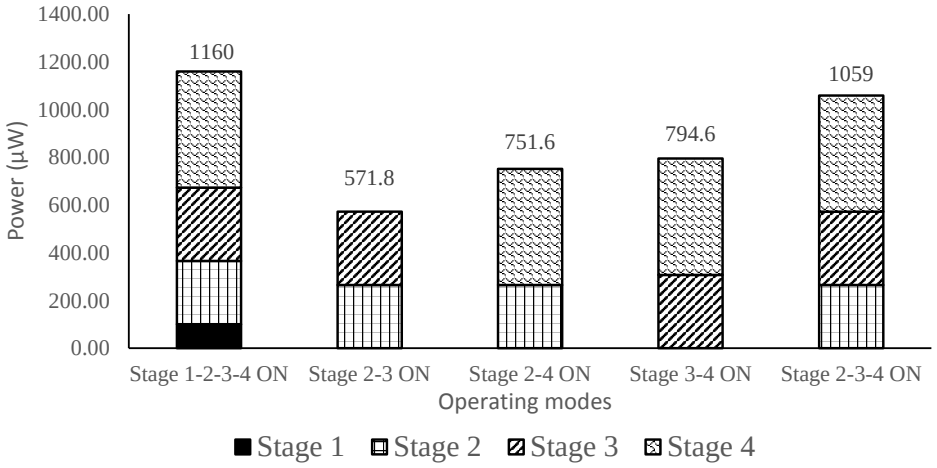


Figure 5.7: Stage-wise power consumption.

architectures, where a bypass of the multiplier unit is established by performing the computation with the second input tied to '1'.

Figure 5.7 shows the power consumption of the proposed DSP architecture for different combinations of DSP pipeline stages. The power consumption can be dynamically tuned by gating different unused stages. When all the pipeline stages are active, the DSP block consumes a maximum of 1160 μW. However it can be observed from the Figure 5.7 that, by clock gating the pipeline stages which are inactive during one clock cycle significant power reduction is achieved. By gating various combinations of pipeline stages a maximum of 50.5% to a minimum of 8.5% power reduction can be achieved. Also, by clock gating stage-3 which contains the multiplier unit, the power consumption can further be reduced by 26.5%. Similar power reduction is achieved by clock gating stages 1, 2, and 4 (8.5%, 22.7%, and 42.0% respectively).

5.3.6 DSP Interconnect

The DSP blocks are usually placed into dedicated columns to facilitate cascading when integrated with SMBs as shown in Figure 5.8. However, being more complex than an SMB, a DSP occupies an area equivalent to a 2×3 array of SMB tiles (or 6

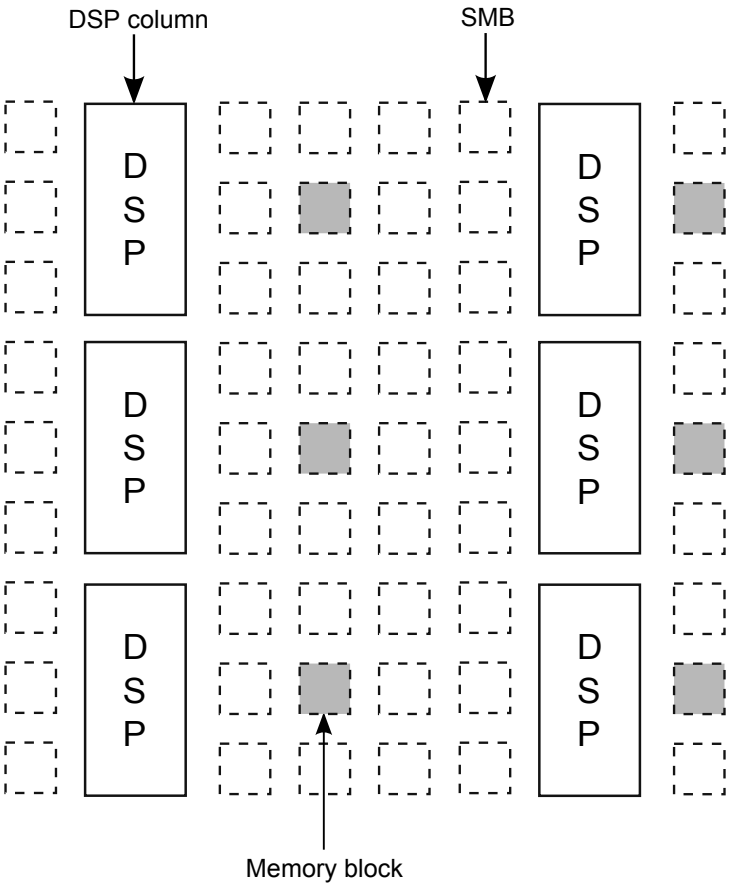


Figure 5.8: Unified architecture with fine-grained and coarse-grained logic.

SMB tiles). The DSP block routing strategy and the associated interconnectivity is the same as discussed in Chapter 4 section 4.3.4 with a mixed wire segment scheme of 20%, 40%, and 40%, for length-1, length-2, and length-4 wire segments. Also the switch box flexibility F_s and connection box flexibility F_c values are set to 3 and 0.25 respectively to provide better interconnect flexibility with general routing structure.

5.3.7 DSP Application - Complex Number Multiplication

Typically the proposed dual pre-adder DSP block is efficient in implementing complex number multiplications which are common in wireless communication systems. One such system is the Software Defined Radio (SDR) in which the digital front end consists of a large number of filter and filter bank structures [208, 209, 210]. For example, a low pass filter with 100 taps requires 100 complex

multiplications. Such multiplication of two complex numbers can be expressed in terms of their individual real and imaginary parts. For a filter with complex input signal, *X(z)* and filter response *H(z)*, its output response *Y(z)* is given by [187]:

$$\begin{aligned} Y_r(z) + jY_i(z) = (X_r(z)H_r(z) - X_i(z)H_i(z)) + \\ j(X_r(z)H_i(z) + X_i(z)H_r(z)) \end{aligned} \tag{5.1}$$

Implementation of (5.1) requires 2 real additions and 4 real multiplications. In general, *4N* real multiplications and *4N-2* real additions are required for a filter of length N [187]. However, an alternative method shown in (5.2) is useful for reducing the arithmetic cost associated with the complex number multiplication. This method requires only 3 real multiplications and 4 real additions to perform one set of complex number multiplication. Hence, for an N-tap filter, the alternative structure requires only *3N* multiplications and *3N+1* real additions per input sample.

$$\begin{aligned} Y_r(z) &= H_r(z)[X_r(z) - X_i(z)] + X_i(z)[H_r(z) - H_i(z)] \\ Y_i(z) &= H_i(z)[X_r(z) + X_i(z)] + X_i(z)[H_r(z) - H_i(z)] \end{aligned} \tag{5.2}$$

By making use of two pre-adders, the pipelined DSP can realize (5.2) in three steps as shown in the Figure 5.9: (*i*) Pre-adder output $\mathrm{X}_r(z)$ - $\mathrm{X}_i(z)$ is multiplied with $\mathrm{H}_r(z)$ and is stored in the intermediate output register. (*ii*) Output $\mathrm{H}_r(z)$-$\mathrm{H}_i(z)$ of pre-adder is multiplied with $\mathrm{X}_i(z)$, stored and added with step (i) to generate the real part of complex multiplication. (*iii*) Pre-adder output $\mathrm{X}_r(z)$+$\mathrm{X}_i(z)$ is multiplied with $\mathrm{H}_i(z)$ which is then added with the value stored in step(ii) to produce the imaginary part of complex multiplication. Using temporal logic folding technique, the complex number multiplication can be performed using only one DSP block. However, the conventional architectures in [5, 80], contain only one pre-adder unit which prevents them from realizing the alternative method in (5.2) in 3 steps. Besides, additional circuitry needs to be incorporated in fine-grained

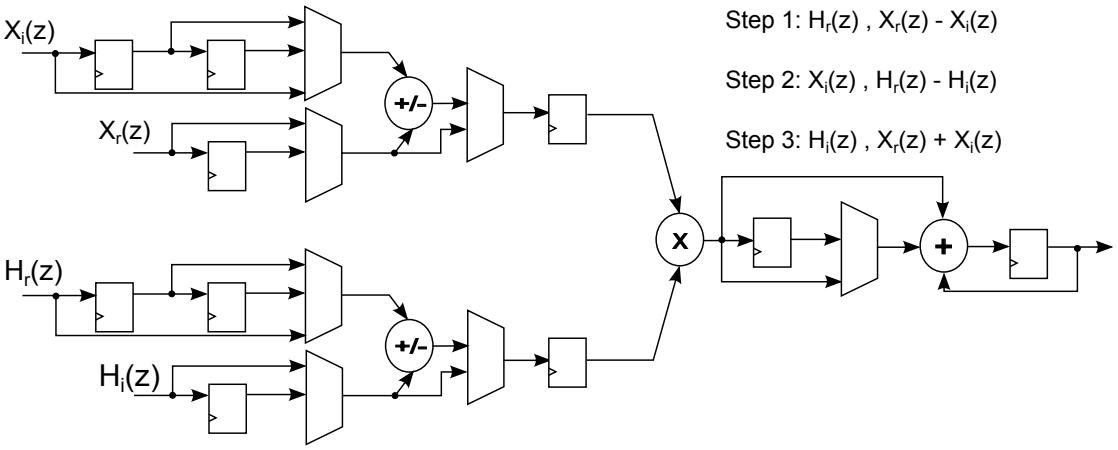


Figure 5.9: Complex multiplication using proposed DSP design.

logic manually by the designer to realize the alternative method, resulting in larger area and power consumption.

In addition to complex multiplications, expressions represented as product-of-sum (POS) can be more efficiently implemented using the proposed DSP architecture over other DSP blocks. For example, realizing the POS expression of the form, $POS = (A + B) \times (C + D)$ using generic DSP blocks with a single pre-adder takes 2 DSP operations. However, the same expression can be realized using only one DSP operation with the proposed pipeline reconfigurable DSP. Hence, for n POS expressions of above mentioned form requires $n - \lfloor \frac{n}{2} \rfloor$ proposed DSP blocks over n traditional DSPs.

However, at a slight area overhead of 5.1% for incorporating the additional pre-adder in the proposed pipeline reconfigurable DSP architecture, efficient implementation of compute intensive kernels is achieved when compared to DSP block with a single pre-adder.

5.4 Extended NanoMap tool flow for Pipelined DSP block

As discussed, the flattened fine-grained NATURE architecture is modified with a pipeline reconfigurable DSP. Modification of NanoMap is hence necessary to fully support the proposed architectural enhancement. During logic mapping the corresponding nodes of an RTL module is implemented with the network of LUTs provided in the module library for specified folding level. The module library is extended with the information for the proposed DSP block. Module parameters like bit width, logic depth, input or output ports, carry ports and configuration bits are all well defined. The initial stages of NanoMap perform logic mapping and scheduling of different nodes (LUTs and DSP).

After assigning LUTs and DSPs into different folding cycles, temporal logic folding enables sharing of hardware resources. The original NanoMap temporal clustering determines which LUTs share the same SMB. Since pipeline reconfigurable DSP enables the sharing of DSP blocks, the clustering algorithm is also modified to perform grouping of DSP blocks to be mapped onto the same physical DSP block. First, all the mathematical operations are directly assigned to individual DSPs. Second, the clustering algorithm considers the connection and life span between DSP operations. Since the DSP supports pipeline reconfiguration, two DSPs with a difference of at least 1 pipeline stage are packed together. The clustering algorithm searches for un grouped DSP blocks and packs them into DSP groups which have fewer shared DSP. This is an iterative process until all the DSPs in the corresponding folding stage are grouped. Further, clustering for the successive folding stages is performed. The DSP blocks that are packed together share the same physical DSP block.

Similarly, the original fine-grained placer and router is modified to support the pipeline reconfigurable DSP blocks. The DSP density can be specified for different architecture and DSP locations are fixed as in the architecture shown in

Figure 5.8. The placer performs swapping of blocks to minimize the delay. However, only blocks of the same type can swap their locations. For example, the SMB cannot be swapped with DSP. Considering routing between DSPs, special vertical interconnects are used to facilitate the chaining of DSPs in the column. Blocks of different types such as DSPs, SMBs and block RAMs are efficiently interconnected using general routing structures. Using the modified VPR tool, total routing delay of the circuit is estimated.

5.5 Results and Discussion

The experimental results for mapping various benchmarks to the proposed pipeline reconfigurable DSP block for the NATURE platform is presented in this section. The reduction in DSP blocks, corresponding area, and power reductions achieved by employing pipeline reconfiguration and logic folding techniques are also demonstrated. Compute intensive benchmarks are like wavelet, Biquad, DCT etc. are used for evaluating the performance of the pipelined DSP design. Of the other bigger benchmarks like ARF, FIR1, FIR2, EWF, HAL, Smooth Triangle and GCD (greatest common divisor), the first 6 benchmark suites are popular DSP applications obtained from UCLAs MediaBench which consists of a large number of addition, multiplication and subtraction nodes [193].

5.5.1 Comparison between Mixed-Grained and Fine-Grained NATURE Architectures

This section explores the A-D trade-offs of mixed-grained (pipeline reconfigurable DSP incorporated NATURE) architecture using different benchmark circuits for various folding levels (FL). These results are then compared to the fine-grained NATURE architecture to show the A-D improvement from incorporating the pipeline reconfigurable DSP block. The A and D parameters in A-D product is the total

Table 5.2: Comparison between mixed-grained and fine-grained NATURE architecture for FL-1.

Benchmark	Fine-Grained			Proposed Mixed-Grained				A×D
	SMB	Total Delay (ns)	Total Area (mm^2)	SMB	DSP	Total Delay (ns)	Total Area (mm^2)	Improvement
ARF	126	44.75	315	0	12	18.25	198	3.90×
DCT	100	48.62	250	2	11	26.18	186.5	2.49×
EWF	73	52.50	182.5	0	15	21.90	247.5	1.77×
FIR1	88	32.50	220	0	8	20.52	132	2.64×
FIR2	68	45.25	170	0	9	21.12	148.5	2.45×
Diffeq	29	72.40	72.5	13	3	52.5	82	1.19×
GCD	8	16.38	20	8	1	73.92	36.5	0.12×
HAL	48	36.00	120	1	3	66.69	52	1.25×
Paulin	21	98.60	52.5	16	4	47.71	106	1.02×
Wavelet	82	42.12	205	16	11	26.67	221.5	1.46×
ASPP4	25	69.90	62.5	16	4	40.26	106	1.02×
Biquad	19	54.04	47.5	8	4	29.52	86	1.01×
Smooth Triangle	82	29.55	29.55	12	10	29.28	29.28	1.06×

Table 5.3: Comparison between mixed-grained and fine-grained NATURE architecture for FL-2.

Benchmark	Fine-Grained			Proposed Mixed-Grained				A×D
	SMB	Total Delay (ns)	Total Area (mm^2)	SMB	DSP	Total Delay (ns)	Total Area (mm^2)	Improvement
ARF	252	27.43	630	0	26	11.22	429	3.59×
DCT	200	28.86	500	3	26	12.33	436.5	2.68×
EWF	146	36.27	365	0	29	11.16	478.5	2.48×
FIR1	175	24.44	437.5	0	20	10.44	330	3.10×
FIR2	135	31.85	337.5	0	22	10.68	363	2.77×
Diffeq	39	43.69	97.5	26	3	27.86	114.5	1.34×
GCD	8	8.77	20	8	1	39.71	36.5	0.13×
HAL	95	23.40	237.5	2	3	35.90	54.5	2.84×
Paulin	42	52.56	105	27	4	22.26	133.5	1.86×
Wavelet	172	23.40	430	35	20	12.06	417.5	2.00×
ASPP4	50	37.95	125	34	6	19.85	184	1.30×
Biquad	32	27.16	80	26	4	16.04	131	1.03×
Smooth Triangle	139	16.32	347.5	12	13	14.68	244.5	1.58×

Table 5.4: Comparison between mixed-grained and fine-grained NATURE architecture for FL-4.

Benchmark	Fine-Grained			Proposed Mixed-Grained				A×D
	SMB	Total Delay (ns)	Total Area (mm^2)	SMB	DSP	Total Delay (ns)	Total Area (mm^2)	Improvement
ARF	500	15.96	1250	0	28	7.68	462	5.62×
DCT	400	22.54	1000	4	26	8.24	439	6.23×
EWF	282	20.93	705	0	34	7.62	561	3.45×
FIR1	348	20.93	870	0	21	7.05	346.5	7.44×
FIR2	268	21.91	670	0	22	7.26	363	5.57×
Diffeq	54	26.55	135	44	3	12.30	159.5	1.83×
GCD	8	9.12	20	8	1	21.78	36.5	0.23×
HAL	188	18.20	470	3	8	18.40	139.5	3.33×
Paulin	64	29.80	160	36	4	11.40	156	2.68×
Wavelet	353	20.93	882.5	67	20	8.10	497.5	4.58×
ASPP4	98	23.20	245	64	7	14.34	275.5	1.44×
Biquad	64	18.32	160	26	4	8.90	131	2.51×
Smooth Triangle	282	11.15	705	12	23	8.22	409.5	2.34×

area (A) and total delay (D) of the circuit. The total area of the circuit is calculated after estimating the area of one SMB and one DSP block using Synopsys Design compiler on a 65 nm TSMC library as target technology. The total area

usage is given by:

$$Area_{Total} = SMB_{Total} \times OneSMB_{Area} + DSP_{Total} \times OneDSP_{Area} \tag{5.3}$$

The critical delay of the circuit is estimated from the VPR tool. The total delay of the circuit is the product of critical delay and total number of folding cycles:

$$Delay_{Total} = Delay_{Critical} \times FoldCycle_{Total} \tag{5.4}$$

$$AD = Area_{Total} \times Delay_{Total} \tag{5.5}$$

The results for FL-1, FL-2 and FL-4 are summarized in Table 5.2, Table 5.3, and Table 5.4 respectively. The tables illustrate various resource utilization such as the total number of SMBs, and DSP blocks for different benchmarks. After estimating the area of one SMB and DSP block and associated routing tracks using Synopsys Design Compiler, the total area usage of different benchmark is determined. The A-D product gain achieved by the mixed-grained over fine-grained NATURE architecture is determined to demonstrate the benefits of proposed DSP block for realizing compute intensive circuits.

In the proposed mixed-grained architecture, DSP blocks are used to realize bigger mathematical operations along the critical paths while smaller computations along the non-critical path are mapped to fine-grained logic. Realizing the benchmark using only fine-grained architecture achieves better area and power optimization at the cost of delay since the area of one DSP block is equal to the area of six SMBs. Also, mapping compute intensive arithmetic operations on DSP blocks can significantly reduce the computation delay. It can be observed across Tables 5.2, 5.3, and 5.4 that as the folding levels change from 1 to 4, the performance (in terms of delay) improves at the cost of an increase in area. Since arithmetic operations are mapped to DSP blocks in mixed-grained NATURE architecture, the total SMB usage is significantly less for compute intensive benchmark suites. An average A-D product improvement of 1.64×, 2.05× and 3.6× is achieved by the proposed DSP incorporated NATURE over the fine-grained NATURE architecture for folding level 1, 2 and 4 respectively. Furthermore, it can be observed that for

bigger benchmarks such as FIR1, FIR2, DCT, Wavelet, ARF, EWF, and Smooth Triangle significant improvement in performance can also be achieved using the pipelined DSP. However, it can be observed that no significant improvement in performance is achieved for smaller benchmark like GCD using proposed DSP block. The critical path delay of the GCD benchmark in both fine-grained and mixed-grained architecture is through the two comparator units which are mapped using LUTs, resulting in large folding cycles and total delay. Hence, for such small benchmark circuits in which the critical path is not along the DSP block, the total delay of the circuit cannot be reduced for folding level 1, 2 and 4. Also, it can be observed from tables 5.2, 5.3 and 5.4 that the total resource utilization for GCD remains unchanged as the maximum number of SMBs across any folding stage is 8, resulting in constant area usage.

To estimate the power consumption of different designs in the mixed-grained NATURE architecture, the power consumption of the DSP block and SMBs are computed using the Synopsys PrimeTime tool, as detailed in [194, 195]. The switching activity file (vcd) is generated across a wide variety of input patterns for both the SMB and the DSP block, which are then provided to the PrimeTime tool to estimate power, using a TSMC 65nm library as the target technology. By estimating the power consumption of these building blocks, the total power consumption of individual benchmark circuits on the unified NATURE is determined as $Power_{Total} = SMB_{Total} \times OneSMB_{Power} + DSP_{Total} \times OneDSP_{Power}$. It can be observed an average P-D product improvement of 10.19× is achieved by mixed-grain over fine-grained architecture across different folding levels.

5.5.2 Comparison between Pipeline Reconfigurable and Full-Block Reconfigurable DSP

Table 5.5 shows the DSP usage comparison of various benchmarks for earlier proposed non-pipelined full-block reconfigurable DSP and pipeline reconfigurable DSP for level-1 folding. It can be observed from the table that pipeline reconfigurable

Table 5.5: Comparison between pipeline reconfigurable and full-block reconfigurable DSPs in NATURE architecture.

Benchmark	Pipeline Reconfig. DSP		Full-Block Reconfig. DSP		% Reduction in		A×D	P×D
	DSPs	Total Delay (ns)	DSPs	Total Delay (ns)	DSP	Area	Improvement	Improvement
ASPP4	4	40.26	6	42.35	33.33	15.34	1.24×	1.25×
ARF	12	18.25	28	23.25	57.14	50.20	2.56×	3.56×
Biquad	4	29.52	4	37.76	0	-11.98	1.14×	1.38×
DCT	11	26.18	32	32.48	65.62	59.40	3.06×	3.93×
Diffeq	3	52.50	3	57.40	0	-9.19	1.01×	1.14×
EWF	15	21.90	34	29.04	55.88	48.74	2.59×	3.60×
FIR1	8	20.52	21	27.30	61.90	55.73	3.01×	4.18×
FIR2	9	21.12	23	27.36	60.86	54.73	2.85×	3.97×
HAL	3	66.69	8	86.45	62.50	55.21	2.89×	3.54×
Paulin	4	47.71	4	50.70	0	-9.50	0.97×	1.11×
Wavelet	11	26.67	24	31.85	54.16	41.83	2.05×	2.08×
GCD	1	73.92	1	78.12	0	-6.73	0.99×	1.08×
Smooth Triangle	10	29.28	37	36.64	72.97	64.89	3.56×	3.43×

DSP based designs require fewer DSP blocks compared to the full-block reconfigurable DSP based designs for most of the benchmark suites. For example, the DCT benchmark requires 32 DSP blocks when mapped using full-block reconfigurable architecture whereas, the proposed pipeline reconfigurable DSP based architecture requires only 11 DSP blocks, resulting in 65.62% reduction in DSP usage. This results in the significant decrement of area and power overhead. For smaller benchmarks like Biquad, Diffeq, Paulin, and GCD, the DSP block utilization is same for both pipelined DSP and full-block DSP incorporated NATURE architecture. This is because all the DSP blocks are scheduled in the same clock cycles in both cases, hence preventing them from further clustering. On an average, 40.33% reduction in DSP block is achieved using the pipeline reconfigurable DSP architecture across 13 benchmarks. Furthermore, the proposed pipelined DSP architecture achieves 31.42% reduction in the area over the full-block reconfigurable DSP architecture. However, no significant reduction in area is achieved for higher folding levels as fewer clusterings of DSP blocks are possible due to resource sharing conflict.

With the significant reduction in DSP resource usage achieved using pipelined DSP blocks, a noticeable improvement in A-D product is also observed across different benchmarks. An average of 2.15× improvement in A-D product is achieved by the proposed DSP architecture over the full-block reconfigurable DSP block.

A significant P-D improvement of up to 2.63× is observed for the proposed pipeline

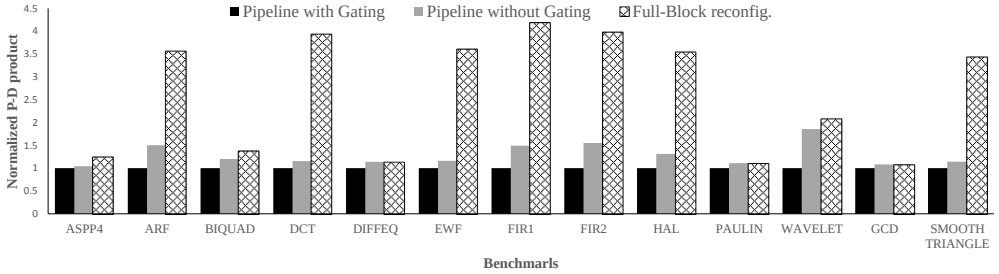


Figure 5.10: P-D product comparison between pipeline reconfigurable and full-block reconfigurable DSP block.

reconfigurable DSP (without clock gating) over the full-block DSP architecture, as illustrated in Fig. 5.10 for level-1 folding. By clock gating individual stages that are unused at any cycle, a maximum P-D improvement of 4.18× can be achieved over the full-block DSP (and 1.85× over the proposed DSP without clock gating) across the different benchmarks. Similar results illustrated in Table 5.5 may be achieved for dynamically reconfigurable architectures that support a full-block reconfigurable DSP, such as FDR [211].

5.5.3 Evaluation of Depth Relaxation on Proposed DSP Incorporated NATURE

From the previous sections, it can be observed that although the resource usage reduction is achieved using the proposed pipeline reconfigurable DSP architecture, some of the benchmarks are not able to fully utilize the full benefit of temporal logic folding. This is mainly due to the as-soon-as-possible scheduling of NanoMap, squeezing many blocks into fewer folding cycles resulting in limited resource sharing. This can be overcome by relaxing the depth (increase in folding cycles) of the circuit which will enable NanoMap to schedule the DSP blocks and SMBs over more clock cycles. Figure 5.11a shows the area-versus-depth relaxation trends for 12 benchmarks for folding level 1. This trend is plotted for depth relaxation values 0, 2, 4 and 6. A rapid decrease in area can be observed with the depth relaxation

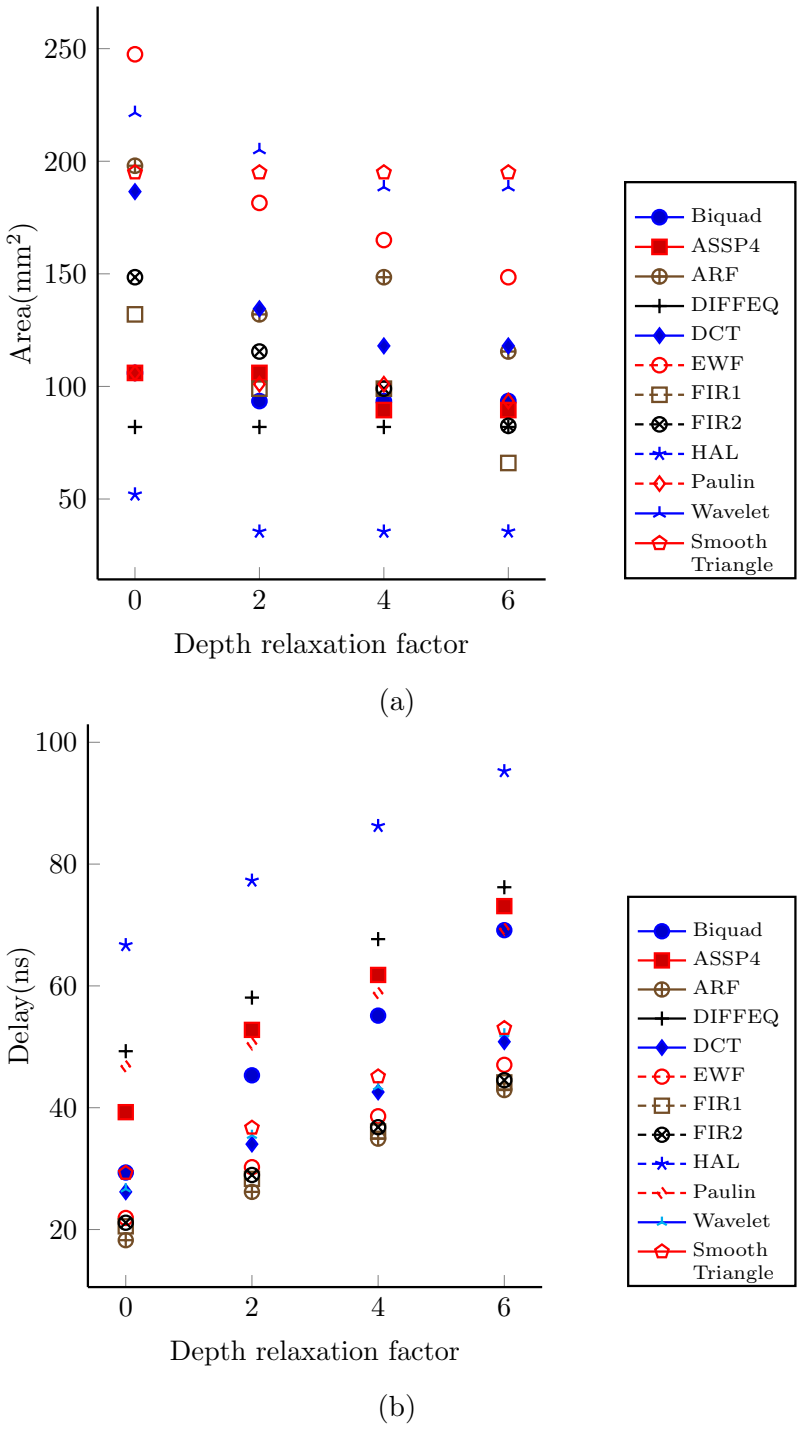


Figure 5.11: (a) Area-versus-depth relaxation. (b) Delay-versus-depth relaxation for FL-1.

for bigger benchmarks like ARF, EWF, WAVELET and FIR. As shown in Figure 5.12a and Figure 5.13a, significant reduction in area can be observed in FL-2 and FL-4 over FL-1, as depth relaxation improves the scheduling of non- overlapping resources at higher folding levels, allowing them to be packed efficiently.

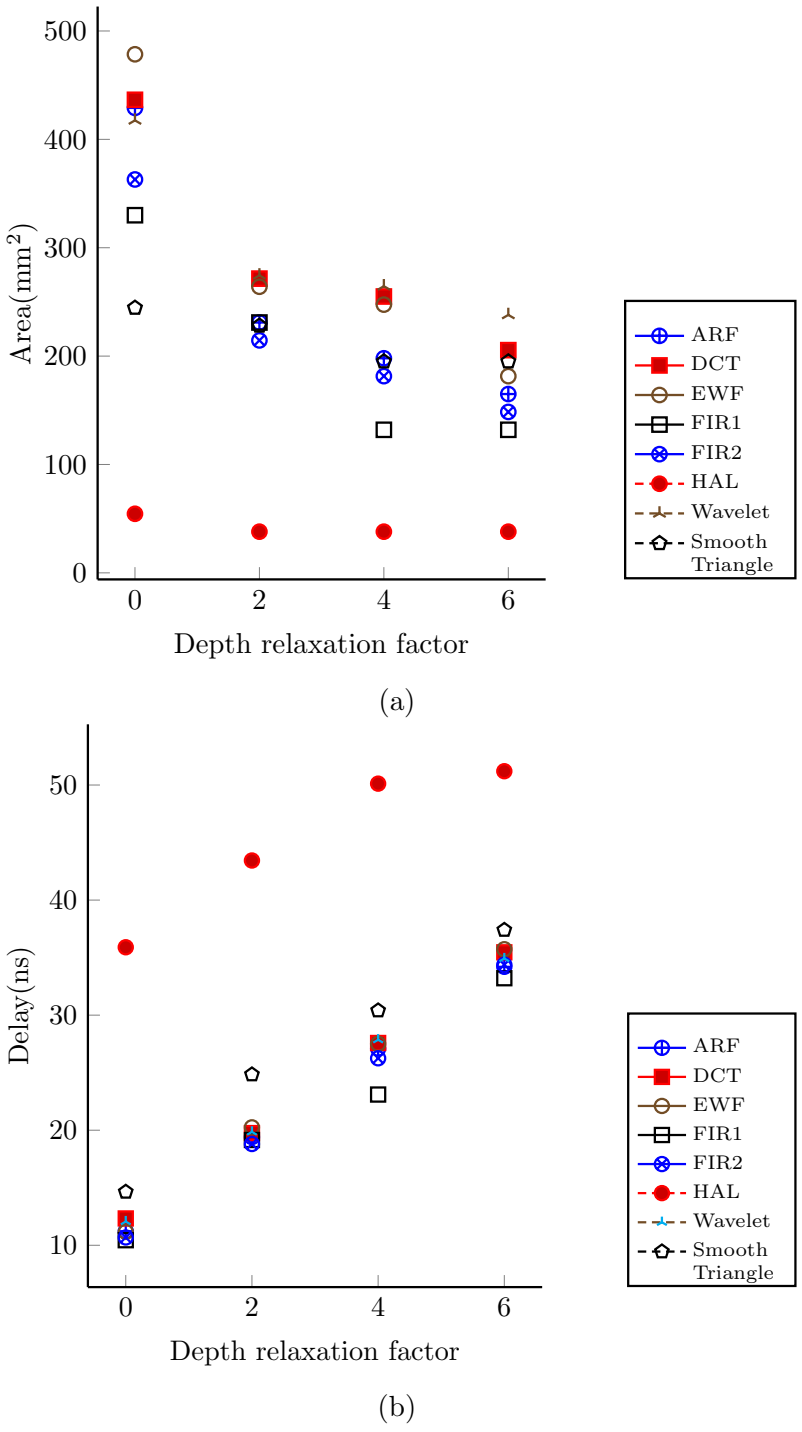


Figure 5.12: (a) Area-versus-depth relaxation. (b) Delay-versus-depth relaxation for FL-2.

However, due to the limited number of DFFs within the DSP blocks, the area improvement tends to saturate for most circuits beyond depth parameter 4. The reduction in area comes at the cost of increased delay. This trend can be observed clearly in Figures 5.11b, 5.12b and 5.13b, which are the delay-versus-depth

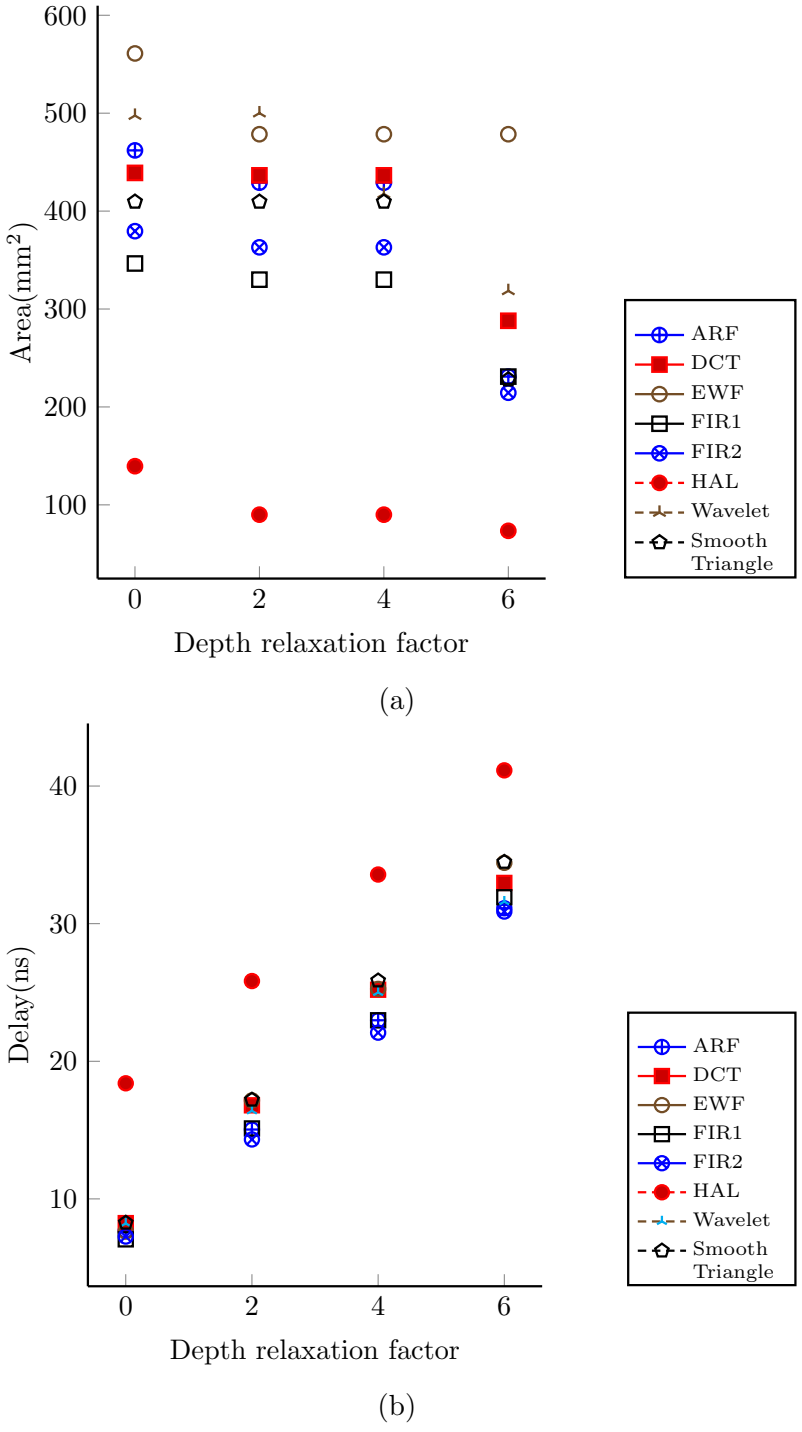


Figure 5.13: (a) Area-versus-depth relaxation. (b) Delay-versus-depth relaxation for FL-4.

relaxation graphs for the 12 benchmarks at folding levels 1, 2 and 4 respectively.

Table 5.6: Comparison between Xilinx FPGA and pipeline reconfigurable DSP in NATURE architecture (FL-0).

Benchmark	Pipeline Reconfig. DSP (90nm)				DSP48A				% Improvement in Area	Speedup
	LUT	DSP	Delay (ns)	LUT_{eqv}	LUT	DSP	Delay (ns)	LUT_{eqv}		
ARF	0	28	5.04	672	0	28	7.20	1792	62.50	1.43×
DCT	14	32	5.99	782	8	29	7.23	1864	58.04	1.21×
EWF	0	34	5.23	816	0	34	7.80	2176	62.50	1.49×
FIR1	0	21	4.74	504	0	21	7.48	1344	62.50	1.58×
FIR2	0	23	4.71	504	0	23	5.74	1472	65.76	1.22×
HAL	37	10	8.70	277	16	10	6.39	656	57.77	0.74×
Paulin	268	4	7.74	364	173	11	10.8	877	58.49	1.40×
Wavelet	269	24	5.71	845	128	28	8.10	1664	49.21	1.42×
ASPP4	430	7	7.57	598	246	7	11.16	694	13.83	1.47×
Biquad	105	4	6.17	201	96	4	6.04	352	42.89	0.97×
Smooth Triangle	12	37	5.21	900	0	37	6.61	2368	61.99	1.27×

5.5.4 Comparison between Xilinx FPGA and Pipeline Reconfigurable DSP in NATURE Architecture

This section provides the details of tests done to compare the performance of the proposed DSP incorporated NATURE architecture for folding level-0 against commercially available FPGAs, using 11 of the same benchmark suites. A Xilinx Spartan-3A DSP is used for a fair comparison with NATURE as the size of LUT on both devices is 4. Since, Spartan-3A family is built on a 90nm process technology, the NATURE architecture is also simulated based on the same technology library. Table 5.6 shows the resource utilization and delay (in ns) for NATURE and Spartan-3A DSP FPGA.

The majority of the benchmarks use the same number of DSP resources. However for the Paulin circuit, in addition to arithmetic nodes, the Spartan-3A DSP FPGA use DSP blocks for control logic as well resulting in increased usage of DSP units compared to NATURE. Across all the benchmarks, an average improvement in speedup of 1.29× is achieved for proposed DSP incorporated NATURE architecture over the Spartan-3A DSP platform.

As DSP blocks and LUTs cannot be compared directly between the two platforms, and to understand overall resource usage, an area comparison in terms of equivalent LUTs is performed. The formula to calculate LUT area equivalent for Spartan-3A

DSP is given by:

$$LUT_{eqv} = nLUT + nDSP \times (64) \tag{5.6}$$

where the area equivalent of DSP48A for Spartan-3A DSP FPGA is 64 LUTs. Similarly, the area equivalent of the proposed pipeline reconfigurable DSP block is 24 LUTs (6 SMBs). From Table 5.6, it can be observed that the proposed pipelined DSP block can achieve a significant improvement in LUT_{eqv} area, resulting in a maximum improvement of 65.76% for the FIR2 benchmark. On average, the proposed DSP incorporated NATURE architecture achieves 54.13% improvement in LUT_{eqv} area over Spartan-3A DSP platform.

5.6 Summary

This chapter presents a pipeline reconfigurable DSP architecture that can further enhance the performance of the NATURE architecture. The proposed DSP block supports independent reconfiguration of pipeline stages to enable efficient utilization of NATURE's logic folding feature, which enables compute intensive tasks such as complex number multiplication to be efficiently implemented using the proposed DSP block. An extended NanoMap tool is also developed to support the proposed DSP blocks.

Simulation results demonstrated that the pipeline-reconfigurable DSP block is able to fully exploit the temporal logic folding capability of NATURE, thereby minimizing the area overhead and increasing the effective logic density utilization. The total power consumption of circuits realized can also be minimized by clock gating the individual pipeline stage of the proposed DSP blocks, with a maximum reduction of 42.0%. Compared with the previous fine-grained NATURE architecture, circuit realization for folding levels 1, 2, and 4 shows that the incorporation of the proposed DSP block can lead to improvements of 1.6×, 2.05× and 3.6× in A-D product, respectively.

Compared to the earlier proposed full-block DSPs based design, the efficient reuse of pipeline reconfigurable DSP blocks enable an average of 31.42% reduction in area and 4.18× improvement in P-D product across different benchmarks. Finally, a comparison between the DSP incorporated NATURE architecture with folding level-0 and Spartan-3A DSP FPGA is also performed. Experimental results show that, on an average, the NATURE architecture achieves an improvement in performance (in terms of speed) of 1.29× and of 54.13% in term of area over the Spartan-3A DSP platform. These results indicate that the proposed reconfigurable DSP design would enable highly efficient implementation of computationally intensive applications when used on the NATURE platform.

6

Fracturable DSP Block for NATURE

6.1 Introduction

Previous chapters discussed how the dynamic reconfigurability of DSP blocks at different levels can efficiently map computationally intensive designs on the NATURE architecture. The full-block reconfigurable DSP block proposed in Chapter 4 supports reuse of resources for realizing arithmetic operations only after its full DSP operation cycles. Similarly, Chapter 5 presented the pipeline reconfigurable DSP block architecture that provides better area and power efficient mapping of arithmetic circuits compared to the full-block DSP on the NATURE architecture. The DSP blocks discussed in earlier chapters incorporate a symmetric 16×16 multiplier block generating a 32-bit result. These DSP blocks are efficient in implementing designs with uniform datapath throughout the circuitry. But it

is common that the input Register Transfer Level (RTL) circuit designs require mixed precision and custom datapath widths to achieve the required accuracy and performance. It has been observed that such mixed precision requirements are usually not efficiently translated to DSP blocks during technology mapping. The fixed precision on these hard DSP blocks often results in sub-optimal utilization, especially in cases where the data-widths are half or less than the input width of the DSP block. In such scenarios, NanoMap will infer a complete DSP block but does not utilize the upper bits for computational purposes, resulting in resource and power wastage. *Variable precision DSPs* have been described in academic research [6, 107] and implemented by commercial vendors (Altera) [197], where DSPs are able to support different computational precision and re-use of resources. However, their precision is fixed for the circuit lifetime during execution, preventing the hard block from being reused within the design for a different operation (across different cycles), which is the key feature provided by temporal logic folding (TLF). Further, vendor tools of commercial FPGAs are currently unable to automatically promote the reuse of DSP blocks based on computational patterns, resulting in inefficient implementation in terms of area and power consumption [212].

In this chapter, a fracturable (variable precision) DSP block architecture for NATURE that can natively support runtime precision selection and temporal logic folding is introduced. The proposed DSP block can switch between sub-width operation mode (2 independent 8×8 operations simultaneously), full width operation mode, or wider multiplication mode (32×32, 24×16 and 24×8 on a single DSP) at runtime. The NanoMap tool is also extended to efficiently map multi-precision arithmetic operations on to the NATURE architecture that incorporates the proposed DSP block. The proposed DSP architecture uses 16-bit operands to be compliant with the NATURE architecture; however, the method can be extended to a 32-bit DSP block that can simultaneously support two 16-bit operations or four 8-bit operations.

The work presented in this chapter is also discussed in,

R. Warrier, S. Shreejith, W. Zhang, C. H. Vun, S. A. Fahmy, *Fracturable DSP Block for Multi-Context Reconfigurable Architectures*, Submitted to Springer Journal of Circuits, Systems and Signal Processing (CSSP), November 2016, pp: 1-14.

6.2 Related Works

Modern commercial FPGAs use highly advanced DSP blocks for accelerating complex computations. Xilinx's 7-series FPGAs use the DSP48E1 [5] architecture, that features an asymmetric multiplier design (25×18) with fixed precision, with a maximum operating frequency of 741 MHz. Altera devices use variable precision DSP blocks that can perform multiple sub-width operations concurrently (up to 3 9×9) with a design-time decision, while also featuring extensions like coefficient memory for efficient implementation of filter blocks [197]. The DSP block supports two precision modes; *i)* Standard-precision mode (18-bit) and *ii)* High-precision mode. In standard precision mode, the DSP block can be configured to support two 18×18 multipliers in either sum mode or the independent mode. The output resolution in later configuration mode is limited to 32-bits. In high-precision mode, the DSP block can be configured to implement a 27×27 multiplier or a 18×36 multiplier. Also, these DSP blocks can be cascaded to further support a wider range of precision modes. However, once the DSP block is configured to support two 18×18 multipliers, it will keep this setting throughout the design thereby not allowing the DSP block to be reconfigured to high-precision mode in runtime.

Academic research on DSP architectures aim to enhance the performance of mixed-precision arithmetic circuits. In [6], a flexible DSP block that enhances multi-input addition operations is proposed. By bypassing the partial product generator into a field programmable compression tree, the DSP block can be configured to perform a multiplication operation or a multi-operand addition. Using this technique, it can perform fused multiply-add operations, or two fused addition-subtraction operations using only one DSP block unlike the traditional DSP blocks

that use two. The area overhead of these DSP blocks is higher due to the usage of fixed-function compression logic and large numbers of multiplexers. A highly versatile DSP block similar to DSP blocks of Altera is proposed in [107] that supports a wide variety of multiplier bit widths as well as multi-input addition with negligible overheads. The proposed technique employs a Radix-4 Booth multiplier architecture, and incorporates two paired 18-bit multipliers in addition to the multi-input addition operation. Through cascading two DSP blocks, it supports eight 9-bit, six 12-bit two 24-bit and one 36-bit multipliers. However, these DSP architectures are primarily designed for single-context FPGAs and hence do not support logic folding.

6.3 Proposed Fracturable Architecture Design

This section presents a fracturable Baugh-Wooley multiplier design, followed by an illustrative example. Further sections explain DSP interconnect design details, power optimization techniques employed and the enhanced functionality of the proposed DSP block for implementing complex number multiplication.

6.3.1 Fracturable Baugh-Wooley (BW) Multiplier with HPM Reduction Tree

The BW multiplier combined with the High-Performance Multiplier (HPM) reduction tree is an efficient way of implementing signed multipliers over other multiplier designs in digital circuits [173]. Parallel multipliers like the BW multiplier operate in three steps: generation of *primary partial products*, *compression* of these multiplier partial products using logarithmic tree and *final addition*. In the BW multiplier architecture, the partial products are generated using an array of AND and NAND gates, which is arranged as shown in Figure 6.1 as per Hatamian's scheme [213] extended for the 16-bit multipliers. These are then reduced using an HPM reduction tree that performs bit-wise compression over multiple stages

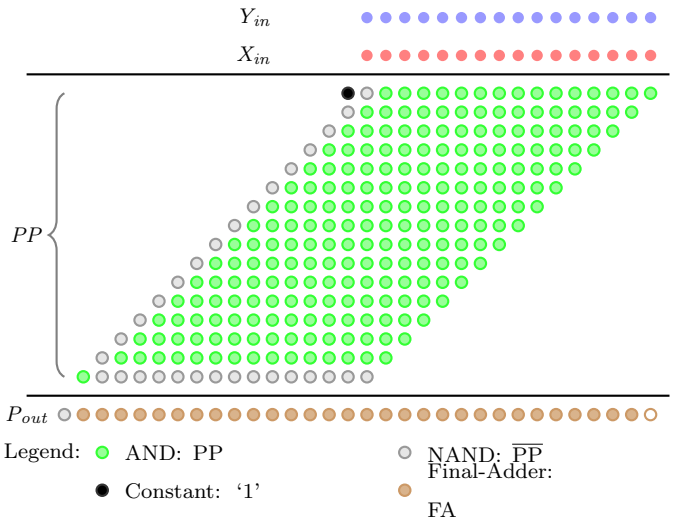


Figure 6.1: Architecture of a 16-bit Baugh-Wooley Multiplier with HPM Reduction Tree.

to reduce the number of partial products, which are then fed to a final adder to compute the product. The final adder employs carry-save adders to perform fast parallel summation of the compressed partial products.

To compute two sub-multiplications in parallel, a fracturing mechanism is introduced at the primary *Partial Product* (PP) generation stage, as represented in Figure 6.2. The multiplier unit uses two configuration bits *mode* and *gate*, that indicate the mode of operation and enable a power saving gating feature, respectively. The illustration depicts the case for a 16-bit multiplier which can be dynamically fractured to perform two 8×8 multiplications simultaneously.

To achieve this fracturing, the proposed architecture makes use of other building blocks to generate the PPs, beyond the regular AND/NAND gates. These are the configurable AND/NAND, Gated-AND, Gated-NAND, Mode-based-Gated-NAND (Gated-Mode-NAND), XOR (Mode-Invert), Mode-OR-AND (Mode-AND) and 1-bit multiplexers (Mode-Mux). For the lower 16-bit partial products, the structure resembles an 8×8 multiplier, with additional rows above and below the regular 8-bit PPs. The partial products are generated by regular AND gates and configurable AND/NAND gates, while the Gated-AND/NAND gates are selectively disabled to allow sub-computations to be performed in isolation. The upper 16-bit PPs also mirrors a similar structure operating on the upper 16-bits

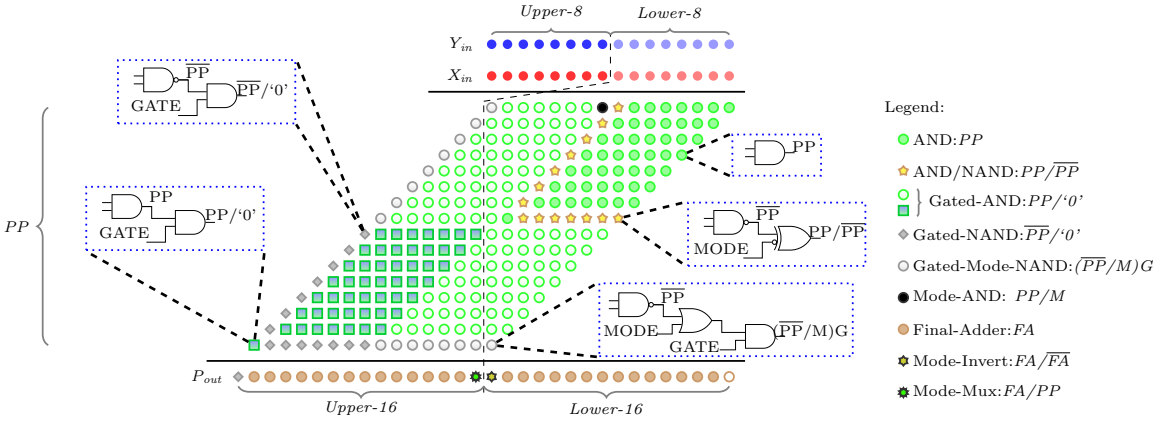


Figure 6.2: Proposed fracturing mechanism of the 16-bit BW Multiplier.

Table 6.1: Operating modes of multiplier.

Gate bit	Mode bit	Operation mode
1	0	Regular 16×16
1	1	Dual 8×8
0	1	Single 8×8

of the partial products. Here, the partial products are generated by Gated-AND, Gated-NAND while the Gated-Mode-NAND generates logic '1' to enable proper combination within the reduction tree. By generating the PP in this manner, the logarithmic reduction tree is reused for all modes of operation, without requiring any changes to its internal structure, reducing the area overhead incurred. The output of the reduction tree is pipelined (single-stage) to limit the critical path within the multiplier.

As shown in Table 6.1, the selection bits *mode* and *gate* determine the three operating modes of the multiplier at any given time. These can be automatically generated by the tool flow (refer 6.5) for altering the modes at runtime. The operation of the circuit in different modes is described below.

(i) Regular 16-bit mode

This mode is selected when *gate* is set to '1' and *mode* is set to '0'. In this case, the circuit falls back to the regular 16 × 16 partial product tree illustrated earlier

in Figure 6.1. With this combination of *gate* and *mode*, the Gated-AND/NAND and Mode-based-Gated-AND/NAND compute regular AND/NAND functions respectively. The configurable AND/NAND gate computes an AND operation while the multiplexers select the first input line, which corresponds to their operation in the regular 16-bit mode. The partial products thus resemble the regular 16×16 partial products, which are then fed to the reduction tree and forwarded to the final adder to compute the product.

(ii) Dual 8×8 mode

This mode is selected when *gate* is set to '1' and *mode* is set to '1'. In this scenario, the lower 8-bits of the inputs X and Y are taken as the input to the *lower-16* section while the upper 8-bits of inputs X and Y are taken as inputs to the upper-16 section. With this configuration chosen, the Gated-AND/NAND computes regular AND/NAND operation, while the Gated-Mode-NAND gates are forced to value '1'. The configurable AND/NAND operates as a NAND gate, while the Mode-AND chooses the value of '1'. This generates two isolated sections of partial products which are then compressed using the HPM reduction tree. The Gated-Mode-NAND gates ensure that the sign bits of the lower-16 bits are unaffected (and contained) during HPM reduction, while allowing the compression of the upper-16 bits in isolation. At the final adder stage, the multiplexers (Mode-Mux) choose the lower bit of the *upper-16* reduced PP, while the Mode-Invert preserves the sign bit of the *lower-16* result.

(iii) Single 8×8 mode

This mode is selected when *gate* is set to '0' and *mode* is set to '1'. In this scenario, the lower 8-bits of the inputs X and Y are taken as input to the *lower-16* section while the upper 8-bits of inputs X and Y are ignored. With this configuration chosen, the Gated-AND/NAND are completely gated in addition to the Gated-Mode-NAND gates, producing an output value '0'. This gating allows

significant power reduction when the multiplication is limited to a single 8-bit scope, compared to the regular 16-bit structure to compute the same. As with the fractured mode, the configurable AND/NAND operates as a NAND gate, while the Mode-AND chooses the value of '1'. Similar to the dual mode, the partial products with *upper-16* gated are then compressed using the HPM reduction tree. At the final adder stage, the multiplexer (Mode-Mux) is also gated to conserve power, while the Mode-Invert preserves the sign bit of the lower-16 result.

The multiplier enables computation of one 16×16 or two 8×8 in full mode, and also a single 8×8 multiplication with reduced power consumption by controlling the *mode* and *gate* pins at runtime. The lower 8-bits of the input pre-adder stages are taken as the lower 8-bits for multiplication in the fractured mode, while the upper 8-bits of the pre-adders are used as inputs to the second multiplier stage (when enabled). Thus, in the fractured mode, the DSP can compute a wider range of computations on 8-bit operands than a standard non-fracturable DSP block, and this flexibility will be explored by the tools for automatically reusing the basic structure.

6.3.2 DSP Block Architecture

The proposed fracturable multiplier is integrated into the previously proposed DSP architecture discussed in chapter 5 to form a complete DSP block that supports both fixed precision and variable precision computations. Figure 6.3 shows the basic block diagram of the enhanced DSP block. This enhanced DSP block contains *gate*, and *mode* pins to reconfigure the BW multiplier in different modes. The selection signals of these control pins are controlled by the multiple configuration bits stored in the associated configuration memory. Also, the interleaved multiplexers allow flexibility to realize different operations to be implemented using this basic structure. The two pre-adders (16-bit each) and the ALU (36-bit) have also been fractured using the same *gate, mode* inputs, allowing four 8-bit add/sub or two 16-bit ALU operations to be performed in addition to sub-width multiplication(s). The modified DSP can compute a wider range of computations

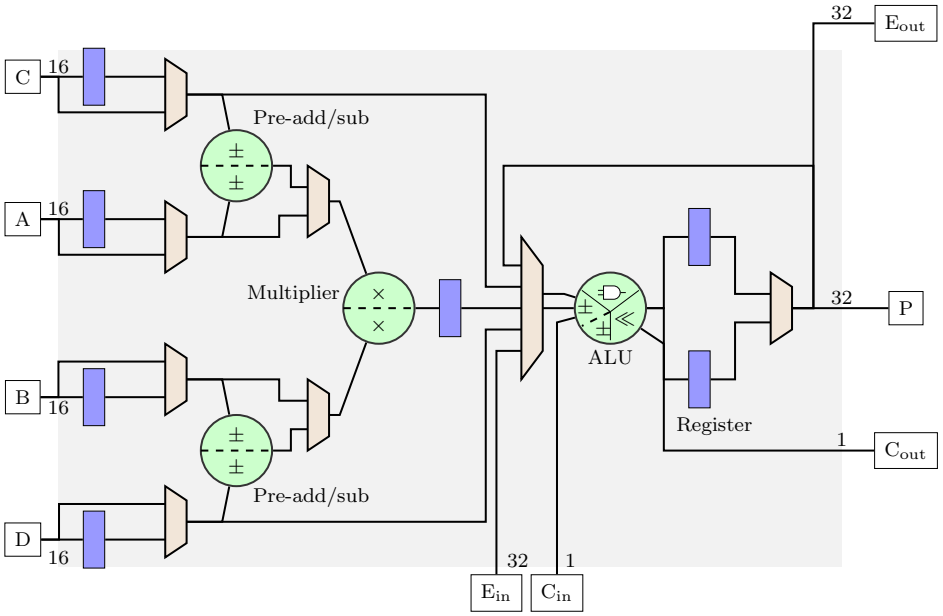


Figure 6.3: A 16-bit Enhanced DSP architecture.

on 8-bit operands than a standard non-fracturable DSP block, and this flexibility will be explored by the enhanced NanoMap.

6.3.3 Supporting wider multiplications

Using TLF, a 32×32 multiplication can be mapped using a single proposed DSP block in 8 clock cycles. The 32-bit operands are separately fed as 16-bit least significant bits and most significant bits over multiple cycles. The partially computed results are stored in the intermediate registers for successive computations. The ALU then performs shift and add operations on the multiplier output to generate the results. Figure 6.4 shows a 32-bit multiplier using only one DSP block with logic folding. Using a similar approach, the proposed DSP block can also implement 24×16 and 24×8 multiplications by reconfiguring one DSP block. Moreover, this approach of realizing wider multiplication takes 1 clock cycle less than the Karatsuba-Ofman algorithm [214], while utilizing the DSP block more efficiently. While folding can be explored for low-power designs, wider multipliers (like 32×32 or higher) for high performance requirements can also be implemented using the Karatsuba-Ofman algorithm through chaining of the DSP blocks.

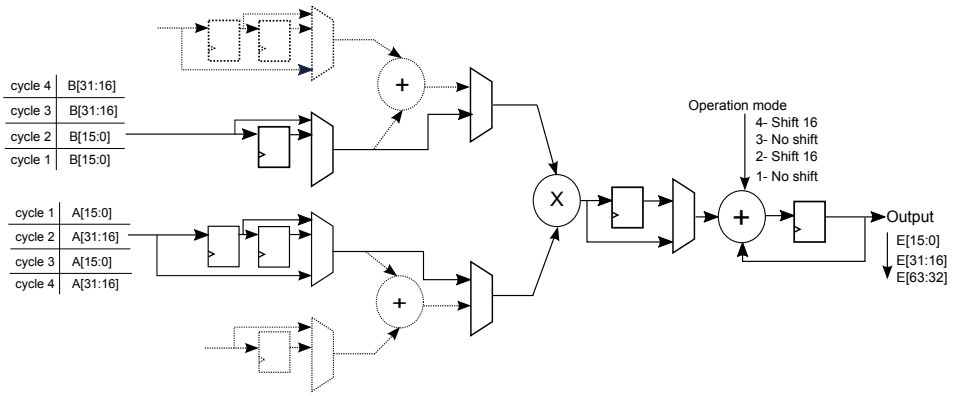


Figure 6.4: 32-bit multiplication using logic folding.

6.3.4 DSP Interconnect

In the proposed DSP block, only the multiplier unit and the corresponding internal datapaths are modified without changing the existing DSP interconnect to support fracturization. Hence the interconnect of proposed DSP block is same as discussed in chapter 5 section 5.3.6.

6.4 Illustrative Example

In this section, the effectiveness of the fracturable DSP block is illustrated, using an RTL circuit that is mapped on to the NATURE architecture.

Consider the scheduled DFG of the circuit shown in the Figure 6.5. It comprises two 8-bit multipliers and an LUT cluster (LUT CLUS-1) scheduled in the first cycle. In cycle two, 2 LUT clusters (LUT CLUS-2 and LUT CLUS-3) are scheduled followed by an 8-bit adder in cycle three and finally, the primary output is derived from LUT (LUT CLUS-4) in cycle four. The input bit width of the circuit is 8-bit and there are intermediate registers after each cycle to hold the output data.

If the circuit is realized using the pipelined reconfigurable DSP block discussed in the previous chapter, it will consume a total of two physical DSP blocks, one each for *MUL1* and *MUL2*. By reconfiguring the pipeline stage of one DSP block, the adder node (*ADD*) scheduled in cycle 3 will be realized. Although the multiplier unit inside the pipeline DSP supports 16×16 multiplication, only the lower 8-bits

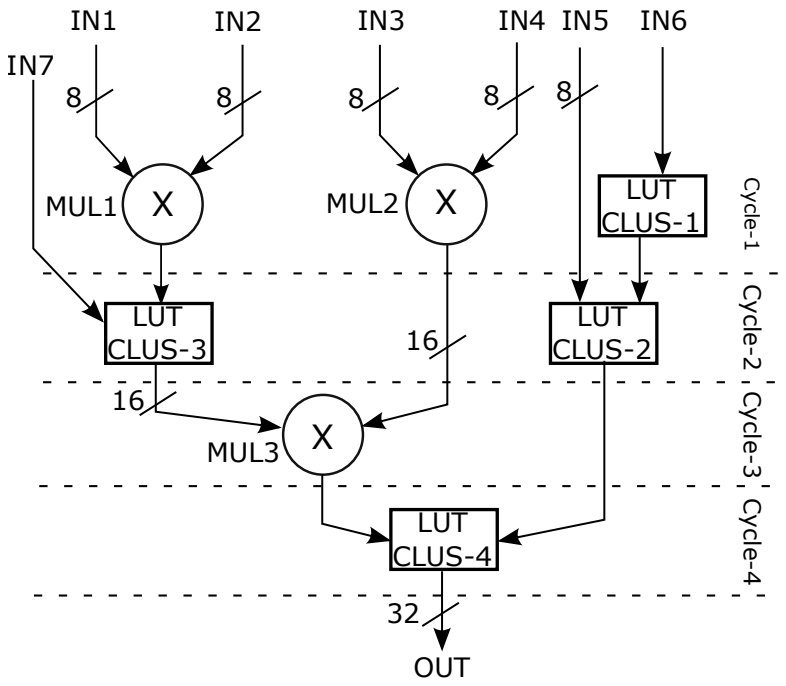


Figure 6.5: DFG of example circuit.

of the multiplier input ports are used since the input precision of the circuit is limited to 8-bits. This leads to the sub-optimal utilization of DSP blocks and results in resource wastage.

By fracturing/splitting the 16 × 16 multipliers block into two 8 × 8 multiplier, both multiplications scheduled in cycle-1 can be realized using a single DSP block. This can be achieved by exploiting the full bit width of multiplier input ports. By careful routing of input signals to be multiplied (*IN1* and *IN2*, *IN3* and *IN4*) to the lower and upper 8-bits of the multiplier inputs, both multiplications can be carried out using a single multiplier unit. Moreover, the same multiplier can be reconfigured to perform 16 × 16 multiplication scheduled in cycle-3. Hence by incorporating the DSP blocks with fracturable multiplier, using only one DSP block, the input circuit can be realized.

6.5 Enhanced NanoMap

The modification incorporated in the DSP architecture requires extending the mapping tool flow. By changing the multiplier design in the DSP block to support

fracturization, it is required to extend the NanoMap tool to cluster the DSP block efficiently by making use of the proposed multiplier capability. Following the logic mapping and scheduling of the input nodes, the clustering algorithm is performed to pack non-overlapping LUTs and DSPs.

In the enhanced NanoMap tool flow, the packing algorithm has been extended to combine DSP blocks with the proposed fracturable multiplier. After mapping the mathematical operations to individual DSP blocks, the algorithm iteratively searches for operations that are scheduled in the same clock cycles. If any two DSP blocks perform sub-width operations (for example multiple 8×8 multiplication) in the same clock cycle (called *mergable DSPs*), the algorithm groups them onto a single DSP block.

The clustering algorithm considers the interconnection and execution cycles of the different DSPs. For this, the algorithm scans through the connectivity graph to identify DSPs which have an operating cycle difference of 1 or more clock cycles (called *affine* DSPs). If each DSP has at least one *affine* DSP, it is marked as *ungrouped* and its list of affine DSPs is saved. Once the algorithm evaluates all the DSP blocks, the list of DSPs in the *ungrouped* list are evaluated and they are merged with their affine DSPs that have minimal cycle gap. Thus the proposed algorithm combines sub-width operations within the same cycle and operations across clock cycles which can be executed on the same DSP block. The pseudo-code for the clustering algorithm is shown in Algorithm 3, and this generates the netlist file for the clustered circuit.

The final step in the tool flow is the placement and routing which is implemented using VPR. The placement and routing has also been modified to accommodate the fracturable nature of primary Inputs/Outputs of DSP blocks.

```
forall clock cycles do
    foreach DSP do
        Check the operation performed
        foreach 8×8 Multiply do
            Identify its mergable DSP
        end
    end
    Combine mergable DSPs
end
foreach DSP do
    forall clock cycles do
        List its affine DSPs if the DSP has atleast one affine DSP then
            mark the DSP as ungrouped
        end
    end
end
foreach DSP do
    if marked ungrouped then
        merge the DSP with the affine DSP
        mark the DSP as grouped
    end
end
```

Algorithm 3: Fracturable DSP grouping algorithm.

6.6 Performance Evaluation and Discussion

The area and power overhead of the proposed fracturable DSP block (previous proposed DSP design) over non-fracturable DSP blocks is presented in this section. The area-delay and power-delay trade-offs achieved using the proposed DSP block are also compared against the previous DSP block in NATURE for both single context and multi-context architectures.

6.6.1 Area/Power Overhead of Fracturable DSP Block

To evaluate the area/power overhead and the reduction in frequency due to the increased logic in the multiplier unit, the proposed DSP block unit is synthesized using Synopsys Design Compiler targeting the TSMC 65 nm cell library. For comparison, the existing DSP block (on NATURE) which uses a standard BW multiplier is synthesized with the same target library. The results are shown in

Table 6.2: Power, area and frequency of the proposed DSP.

Design	Cell Area	Dynamic Power(μW)			F_{Max}
	(μm^2)	One 8×8	Two 8×8	Full 16×16	(MHz)
Non-Fract. DSP	16487	934	–	1160	333
Fract. DSP	18183	955	1140	1191	400

Table 6.2. The pipelined multiplier in the proposed DSP block results in a higher operating frequency (F_{Max}) of 400 MHz, with 10.2% increase in cell area over the existing (non-fracturable) DSP block.

The power consumption of the proposed DSP block is estimated using the Synopsys PrimeTime tool. The activity file was generated for large number of input patterns for each operating mode, which was provided to the PrimeTime tool for estimation. It is observed that with fracturing, the proposed architecture results in a 38.3% reduction in power and 43.8% reduction in area when two 8-bit multiplications are scheduled in parallel, which utilizes two 16-bit multipliers with an existing DSP block in NATURE. For single 8-bit and full precision modes, operating at 400 MHz results in a slight increase in power consumption of 2% and 2.67% respectively over the previous DSP block in NATURE that runs at 333 MHz. At 333 MHz, the fracturable design consumes 15% less power for both single 8-bit and full precision operations, compared to the existing DSP block. Thus a clear advantage is achieved in terms of performance and power consumption for the proposed DSP structure in mixed precision digital circuits, which are commonly used in many applications found in audio, vision systems and others. The advantage is further demonstrated using actual circuits in the following section.

6.6.2 Experimental Results on Single & Multi-context Architectures

The experiment is performed using generic RTL/netlists of circuits for Discrete Cosine Transform (DCT), Auto-regression Filter (ARF), Application-Specific Programmable Processor (ASPP4) [192], Greatest Common Divisor (GCD), Paulin

(Differential-equation solver) [191], Wavelet Transform, and Finite Impulse Response filters (FIR1 and FIR2). In addition to these generic circuits, the evaluation of the proposed DSP is also performed using complex functions such as the Elliptical Wave Filter (EWF), HAL, Smooth Triangle, HornerBezier, Motion Vector, and Matrix Multiplication. The input precision is set to 8-bit for all evaluations. The evaluation of A-D and P-D trade-offs is performed using the proposed fracturable DSP block for single-context (folding level 0) and multi-context (folding level 1) NATURE architecture, and are compared against the DSP block proposed in previous chapter.

(i) Evaluation on Single-Context Architecture

This sub-section explores and quantifies the reduction in area and power obtained using the fracturable DSP over the non-fracturable DSP block for folding level 0. The results of folding level 0, which can be considered equivalent to a single context FPGA is also compared against the results based on an Altera Stratix V (5SGSMD4E1H29C1) device that features 6-input fracturable LUTs and variable precision DSPs. These comparison results are presented in Table 6.3. To account for the difference in platforms, the effective area utilized by the implementation (in terms of equivalent LUTs) is computed using the relation $A_{Eff} = LUT_{Max}/DSP_{Max} * DSPutilization + LUTutilization$ [215]. In addition, the number of multiply/MAC and add/sub operations in each benchmark are also shown (in brackets), which helps in determining the reduction in DSP blocks achieved by exploiting their fracturable nature (on Stratix V and on the proposed DSP in NATURE). The A×D improvement of the proposed DSP architecture is compared against the previous DSP structure, for the different benchmark circuits indicated earlier. The table also illustrates the resource utilization in terms of the total number of CLBs, and DSPs, total delay (in ns) and the area consumption of the different benchmarks, for the fracturable and non-fracturable DSP blocks. Furthermore, it also presents the % reduction in area and DSP usage achieved with the proposed fracturable DSP for all the circuits.

Table 6.3: Resource comparison of benchmark circuits implemented on NATURE (with fracturable DSP block and with non-fracturable DSP block) for folding level-0 and an Altera Stratix V 5SGSMD4E1H29C1.

Benchmark (no of mult/mac, add/sub ops)	Previous DSP		With Fract. DSP				% Reduction		Altera Stratix V			A_{Eff}
	DSPs	Min. Period	DSPs	LUTs	A_{Eff}	Min. Period	DSP	Area	DSP	LUTs	A_{Eff}	Improvement
DCT (13,19)	32	4.12 ns	13	16	1680	3.22 ns	59.38	50.79	12	146	1707.4	1.02×
ARF (16,12)	28	3.79 ns	10	0	1280	2.92 ns	64.29	57.86	12	54	1615.4	1.26×
FIR1 (11,10)	21	3.57 ns	8	0	1024	2.94 ns	61.90	55.05	11	126	1557.3	1.52×
FIR2 (8,15)	23	3.61 ns	8	0	1024	2.88 ns	65.22	58.96	8	122	1162.9	1.13×
Wavelet (10,14)	24	4.09 ns	12	144	1680	3.37 ns	50.00	31.50	10	184	1485.2	0.88×
ASPP4 (3,4)	7	5.08 ns	6	224	992	4.35 ns	14.29	-0.44	3	148	538.4	0.54×
EWF (8,26)	34	3.75 ns	14	0	1792	3.27 ns	58.82	51.41	8	210	1250.9	0.70×
HAL (6,4)	10	6.06 ns	4	40	552	5.09 ns	60.00	43.96	6	46	826.7	1.50×
Paulin (2,2)	4	4.68 ns	3	140	524	3.79 ns	25.00	4.17	1	84	344.3	0.65×
HornerBezier (8,3)	11	3.74 ns	4	12	524	3.32 ns	63.64	54.12	8	50	1090.9	2.08×
MotionVector (12,12)	24	3.51 ns	6	16	784	2.83 ns	75.00	68.21	12	146	1707.4	2.17×
MatrixMult (48,12)	60	4.72 ns	32	96	4192	3.46 ns	46.67	34.31	48	482	6729.5	1.60×
Smooth Triangle (17,20)	37	4.13 ns	17	48	2224	3.25 ns	54.05	42.98	17	194	2405.9	1.08×

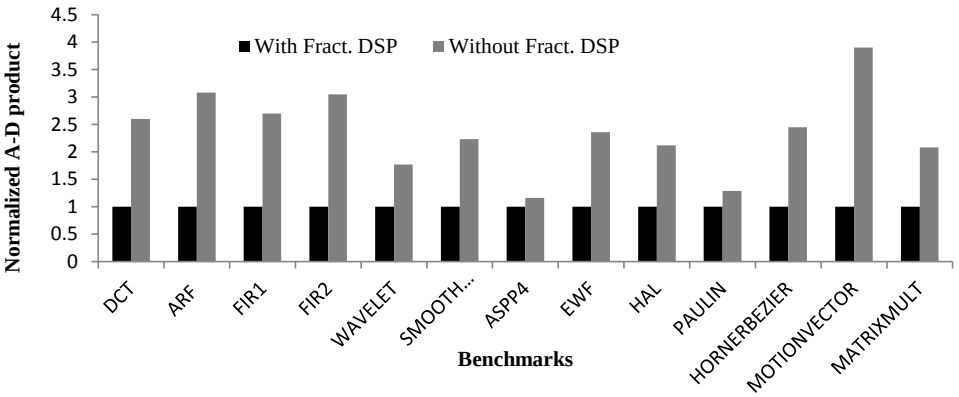


Figure 6.6: A-D product comparison between fracturable and non-fracturable DSP blocks.

Among the 13 benchmarks, the % reduction in DSP usage is significant for all multiplier dominated circuits such as ARF, FIR1, FIR2, Wavelet, HornerBezier, Matrixmult and HAL. Here, two 8-bit multipliers scheduled in the same clock cycle are combined to a single fracturable DSP unit, which is impossible to achieve in the case of a non-fracturable DSP block, resulting in reduced DSP utilization. This is shown in Table 6.3 where the proposed DSP block achieves significant reduction in DSP utilization compared to the previous fixed precision DSP (non-fracturable) architecture in NATURE. This is achieved by merging the two sub-width mult/MAC or add/sub operations that are scheduled in the same clock cycle into a single DSP block compared to two DSP instances with the previous DSP architecture. This results in an average reduction of 53.7% in DSP utilization, and 42.5% in area, across all benchmarks. Figure 6.6 shows the normalized A-D product comparison between the two implementation platforms (fracturable DSP-based FPGA and previously proposed DSP-based FPGA) for the different benchmark circuits. Although a minor increase in routing delay is incurred when merging two 8×8 multipliers, the proposed DSP is able to be clocked at a higher frequency resulting in lower circuit delay. This results in an overall reduction in the A-D product. It can be observed that when compared to a non-fracturable DSP, an average A-D improvement of 2.56× is achieved by the fracturable DSP based reconfigurable architecture across all benchmarks.

Figure 6.7 illustrates the normalized P-D product comparison between the two

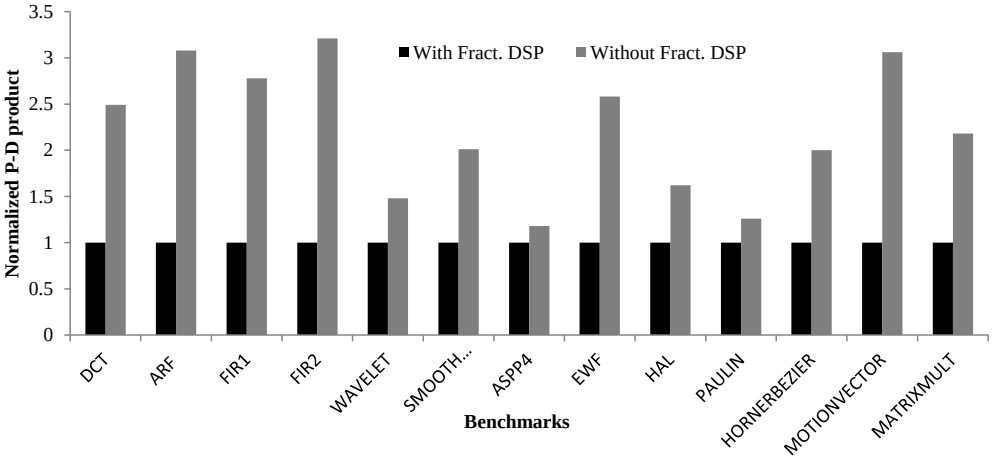


Figure 6.7: P-D product comparison between fracturable and non-fracturable DSP blocks.

implementation platforms for the different benchmark circuits. The gating operation allows multiple computations to be performed in isolation on the same block, leading to lower DSP usage and reduced power consumption. Furthermore, the gating logic allows single 8×8 and dual 8×8 computations to be more power efficient on the proposed DSP compared to non-fracturable DSP block. These translate to an average P-D product improvement of 2.23× on the proposed DSP based FPGA across all the benchmark circuits.

Implementation on Altera Stratix V device maps only mult/MAC operations onto DSP blocks while add/sub operations are implemented using LUTs. The proposed fracturable DSP achieves an average reduction in effective area of 1.24× (across all benchmarks), despite the Altera device having superior LUT (6-input fracturable vs 4-input fixed on NATURE) and DSP (three 9×9 v/s two 8×8 for the proposed DSP on NATURE) architecture. It can also be observed that the Quartus tool automatically merges sub-width operations to reduce DSP utilization in multiple benchmarks (DCT, ARF, PAULIN); however, the enhanced NanoMap tool-flow is able to further reduce the DSP utilization by exploiting the fracturable Pre-add/sub and Post-add/sub blocks in the proposed DSP.

Table 6.4: Efficiency improvement for fracturable DSP for folding level-1.

Benchmark	Non-Fractuable DSP		Fracturable DSP		A×D	P×D
	DSPs	Min. Period	DSPs	Min. Period	Improvement	Improvement
Wavelet	11	3.81 ns	8	3.14 ns	1.45×	1.23×
ASPP4	4	3.66 ns	2	3.68 ns	1.49×	1.08×
Paulin	4	3.67 ns	3	3.4 ns	1.23×	1.08×
Motion Vector	6	3.62 ns	5	3.01 ns	1.27×	1.15×
MatrixMult	23	3.69 ns	21	3.58 ns	1.01×	1.03×
Smooth Triangle	10	3.65 ns	9	3.36 ns	1.07×	1.02×

(ii) Evaluation on Multi-Context Architecture

Table 6.4 shows the mapping results (DSP utilization, critical delay, A-D, and P-D products) of the benchmark circuits on NATURE architecture with the proposed DSP blocks and on NATURE with non-fracturable DSP blocks at folding level 1. Here, the logic is folded (reconfigured) at a depth of 1 LUT computation. In this mode, apart from combining two 8-bit multipliers scheduled in same clock cycle to a fractured DSP, it can be reused across clock cycles to implement subsequent operations if there are no resource conflicts. Furthermore, each DSP block may vary its configuration from full 16×16 mode to the power-saving single 8×8 mode or a fractured dual 8×8 mode across different cycles, as determined by its configuration bits. This flexibility allows further optimizations in resource consumption, compared to folding level 0, which was discussed earlier in Table 6.3.

In Table 6.4, the 6 benchmarks which offered reduced resource consumption between the two target platforms (NATURE with non-fracturable DSP block and NATURE incorporating proposed DSP block) is shown. For each of the benchmark, it can be observed that the overall resource consumption was reduced in folding level 1, with both DSP blocks. An average reduction of 22.93% in DSP utilization across the 6 benchmarks and an average A-D improvement of 1.25× is achieved for NATURE architecture that incorporates the proposed DSP block. For other benchmarks (FIR1, FIR2 and others), it can be observed that temporal folding introduces resource conflicts and limits the scope of DSP reuse through fracturing.

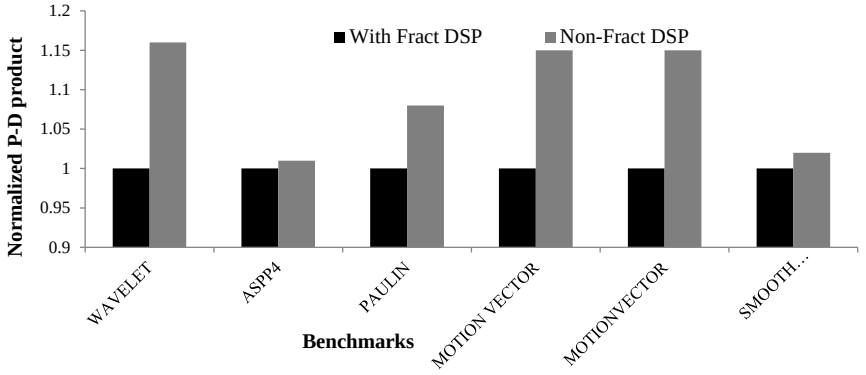


Figure 6.8: P-D product comparison between fracturable and non-fracturable DSP blocks for block processing applications.

Figure 6.8 shows the normalized P-D product for the circuits implemented on the fracturable and non-fracturable DSP block. By efficient merging and reuse of the fracturable DSP blocks, it is able to further reduce the DSP resource consumption over the non-fracturable DSP, resulting in improved area utilization and reduced power consumption. It can be observed that an average P-D product improvement of 1.09× is achieved by mapping applications on fracturable DSP over the existing DSP block on NATURE architecture for folding level 1. Based on the results achieved for the benchmark suite discussed, it can be deduced that the proposed DSP block can result in more significant improvement in the energy efficiency (in terms of P×D) without compromising system performance for circuits with higher computational complexity.

6.7 Summary

This chapter introduced a fracturable DSP architecture for improving energy efficiency of computations on FPGAs. The proposed DSP block achieved this efficiency by fracturing its internal data-path while also providing the capability to dynamically switch computation precision. By utilizing this capability, the proposed DSP block can efficiently handle two independent half-width (8×8) multiplications in complete isolation, perform a single 8×8 multiplication with lower power consumption or operate on regular full-width 16-bit operands. Furthermore,

these modes can be switched dynamically, allowing efficient reuse of the DSP block for low-power applications. The NanoMap tool flow have been extended to take advantage of this flexibility to automatically promote reuse of the DSP blocks by analyzing the computational pattern of the input circuit. Experimental results show that mapping benchmarks circuits onto the NATURE architecture that incorporates the proposed fracturable DSP block achieved 42.5% and 53.7% average reduction in area and DSP block utilization with 2.23× improvement in energy efficiency without utilizing temporal folding (folding level 0). Further improvement in energy efficiency and resource utilization is achieved when temporal folding is employed, without sacrificing performance.

7

Area/Power-Aware NanoMap

7.1 Introduction

As shown in the previous chapters, a significant reduction in area and performance overheads can be achieved by implementing compute intensive kernels on hybrid reconfigurable architectures through the choice of appropriate DSP configurations and precision modes, together with judicious choice of folding level.

However, the optimum setting was determined manually by exploring all the possible configurations. It requires the designer to run the NanoMap tool for all the 16 different configurations. As there are 4 folding levels and 4 DSP block configurations per folding level, a total of 16 configurations are tested and compared in order to determine the best configuration (in terms of DSP configuration and

folding level) for minimum A-D product or P-D product value for each given input constraint. This is a tedious process which consumes a considerable amount of time for each iteration, and thus limits NATURE from being used for fast hardware prototyping. This chapter hence presents optimization techniques that can automatically determine the optimum folding levels and DSP modes by estimating area/power trade-off without implementing all configurations. The most efficient mapping of the chosen configuration is subsequently fed to the mapping flow (NanoMap) to generate the bitstream.

The following section of this chapter discusses some of the previous works on low power and area optimization techniques employed in architecture and mapping tool level. An automated sub-optimal depth relaxation algorithm is then presented, that aims to minimize the area utilization of NATURE architecture without degrading the performance of system, hence achieves optimum A-D value. The chapter also discusses various optimization techniques that can be used in the mapping tool to rapidly determine the best energy (P-D product) configuration. The results and discussion section then demonstrates the benefits of proposed A-D and P-D optimization techniques.

The work presented in this chapter is also discussed in,

> R. Warrier, W. Zhang, C. H. Vun, *Power-Aware High-Level Synthesis for Dynamically Reconfigurable Architectures*, IEEE Transaction on Computer-Aided Design (TCAD) (prepared for submission).

7.2 Related Works

The long design cycles due to rising complexity of applications and architectures offer a major challenge to the circuit designers to meet the required A-D constraints while implementing efficient systems. Generally, the designers perform design space exploration through several iterations of synthesis flow to reach a good constraint compliant solution. Exploring and choosing the best solution that

matches the constraints of application usually is a time consuming process when done manually.

Energy efficient FPGA implementation has become a key performance metric in the design of various computation and communication systems. It is critical for wireless applications and computationally intensive kernels operating on portable hardware platforms. However, such application mapping on custom reconfigurable platform generally suffers from poor energy efficient implementation because the designers ends up giving importance to improved performance and area reduction. Hence a design space exploration and optimization techniques for energy efficient application mapping on custom reconfigurable architectures is required.

Design space exploration is the process of analyzing several *functionally equivalent* implementation alternatives to identify optimal solutions. Earlier works on low-power design space explorations for FPGAs focus on architecture level mainly, register binding, functional unit binding and resource allocation problems [216, 217]. Other techniques include mapping input design using coarse-grained blocks than fine-grained configurable blocks in FPGA, since the former are more power efficient that the latter for the same function [125]. For streaming applications like FIR filters, cosine transforms, pipelining techniques are used prominently for reducing the glitches and hence minimizing power consumption [126]. Also word-length optimization [218], clock gating [128], and dynamic voltage scaling [219] are some of the architecture level power optimization techniques employed in the literature.

Over the period of several years, researchers have worked on architecture level and mapping tool level optimization of FPGA implementation to estimate area-delay trade-offs. In [137], area-delay estimation is determined based on a library of benchmarks implemented and characterized for several FPGAs. The main bottle-neck of this approach is the requirement of different libraries for various task for diverse devices and applications. Based on Xilinx XC4000 device, Xu. *et al.* proposed a technique to compute area and delay values from an estimation obtained from mapping and place & route stages [138, 139]. Techniques proposed in [140]

estimate area and delay at abstraction level (DFG) in which a combination of both algorithm characterization and FPGA mapping model is used. However, it fails to consider the control and multidimensional data overheads. An iterative compilation based method is proposed in [142], which performs extensive transformation, approximation and optimization of DFF nodes that are mapped onto FPGA to compute the best area and delay values. [144] is another compiler-based technique that estimate area and delay based on the synthesis results of a high-level synthesis tool. Choi *et al.* proposed a technique suitable for a large design space to efficiently estimate the area-delay values [146]. However, most of the research works focused on a single RTL architecture that proposes several implementation models.

Power optimization techniques for FPGA architecture are also applied to CAD tools which map the input design to the reconfigurable fabric. Power-aware high level synthesis algorithms for FPGA are proposed in [147, 149]. Moreover, power optimization is also applied on different stage CAD tools such as technology mapping, clustering, placement and routing stages [150, 157, 160]. Although, research on power optimization techniques for FPGA have been carried out for years on different levels, energy efficient mapping of applications for a reconfigurable architecture like NATURE that provide varied configuration levels has not been widely studied.

7.3 Proposed Area/Power-Aware Algorithms for NATURE

In this section, a heuristic approach to determine the optimum A-D product configuration to map a compute intensive kernel on multi-context FPGA architectures along with a design example is presented. Further sub-sections explain various optimization techniques that can be used in the NanoMap tool to readily determine the best energy configuration.

7.3.1 Area-Delay Minimization using Sub-Optimal Depth Relaxation

In NanoMap, various circuit parameters such as logic depth, width of logic in each plane, number of planes etc., determine the number of folding stages, and subsequently affect the scheduling of LUTs and LUT clusters.

The logic depth of a circuit is defined as the number of LUTs along the critical path of the circuit. NanoMap uses an ASAP scheduling algorithm to compute the logic depth. Starting with *depth* parameter variable set as one, the ASAP algorithm traverses through the input edge of the first node and assigns it with an initial depth value (one). It then moves to the next node that is connected to the output of the previous node, and increases the value of the *depth* variable by one. This process repeats until the algorithm reaches the final output edge. The final value of the variable provides the maximum depth of the circuit. After determining the depth of the circuit for a given folding level, the folding stage is computed as follows:

$$\#total\_folding\_stage = \left\lceil \frac{logic\_depth\_max}{logic\_folding\_level} \right\rceil \tag{7.1}$$

where logic_depth_max is the maximum depth of the circuit across all the planes.

Once the folding stages are determined, the tool uses FDS for the assignment of LUT and LUT clusters into different folding cycles, aiming to balance the resource usage across computed folding stages in each plane. Since the architecture is designed to support deep logic folding, it can be deduced that the total number of folding stages will be equal to the logic depth of the circuit. Using an iterative approach followed by the FDS method, each LUT and LUT cluster can be assigned to different available folding cycles of minimum force.

After scheduling, LUTs and LUT clusters are packed into LEs to simplify the placement and routing. As the folding level is set to the lowest possible value,

further minimization in area is possible only by increasing the logic depth. This will increase the number of folding stages such that the LUT and LUT cluster can be assigned to a greater number of folding cycles. This enables more resource sharing with the penalty of a longer delays in the circuit. As such it is necessary to provide a means to control the amount of delay by limiting excessive depth relaxation.

An adaptive algorithm is proposed in this chapter to determine the optimal depth for minimum area under a set of limiting parameters.

The proposed algorithm uses an iterative approach to obtain the minimum LE resource usage. It consists of two parts - one is to determine the optimized depth for minimum area for a given delay constraint and the other is to get the minimum area without delay constraint.

For delay constraint optimization, the algorithm determines the delay of the circuit (*Cur_delay*) and compares it with the input delay (*Sample_delay*). If the delay (*Cur_delay*) is less than *Sample_delay*, the depth is relaxed by one step and the process repeats. During each iteration, after clustering, the LE resource usage *Cur_LE_Count* is determined and is stored along with its corresponding relaxed depth. When *Cur_delay* exceeds the *Sample_delay* the process is stopped. The algorithm selects the depth which has the minimum value of LE_count. Using the obtained depth, the LUT and LUT clusters are scheduled, and grouped to generate a netlist for further processing.

The next step is to determine the optimized depth for minimum area identification without delay constraint, the algorithm searches for minimum LE_count. During each iteration, the algorithm checks whether *Cur_LE_count* is less than or equal to *pre_LE_count*. If the condition satisfies, then *Depth_max* is relaxed by step one and LUT scheduling and clustering is performed. This process repeats until the condition is violated and it generates the netlist using the identified minimum LE_count. The algorithm also checks for the consistency of LE usage during each depth relaxation stage. If the LE_count value remains the same for some iterations, the algorithm terminates searching for minimum LE resource and fixes it with the

```
cur_LE_count =0;
Count =0;
if delay_constraint == TRUE then
    if Cur_delay < Sample_delay then
        Schedule LUT and LUT_cluster;
        Temporal_Cluster LUT and LUT_cluster;
        Depth_max = Depth_max+1;
        Store [Cur_LE_count, Depth];
    else
        Search for minimum LE_count and depth from stored value;
        while Depth minimum do
            Schedule LUT and LUT_cluster;
            Temporal_Cluster LUT and LUT_cluster;
            Generate net_list;
        end
    end
else
    if cur_LE_count <= pre_LE_count then
        if cur_LE_count = pre_LE_count & Count <=3 then
            Schedule LUT and LUT_cluster;
            Temporal_Cluster LUT and LUT_cluster;
            Depth_max = Depth_max+1;
            Count = Count+1;
        else
            Schedule LUT and LUT_cluster;
            Temporal_Cluster LUT and LUT_cluster;
            Count = Count+1;
            Count =0;
        end
    else
        Generate net_list;
    end
end
```

Algorithm 4: Depth relaxation algorithm.

constant value to reduce the search timing. Through experiments conducted on various benchmark suites, the maximum number of iterations to be performed to check the LE usage consistency is identified to be 3. The proposed algorithm is summarized in Algorithm 4.

7.3.2 Design Example

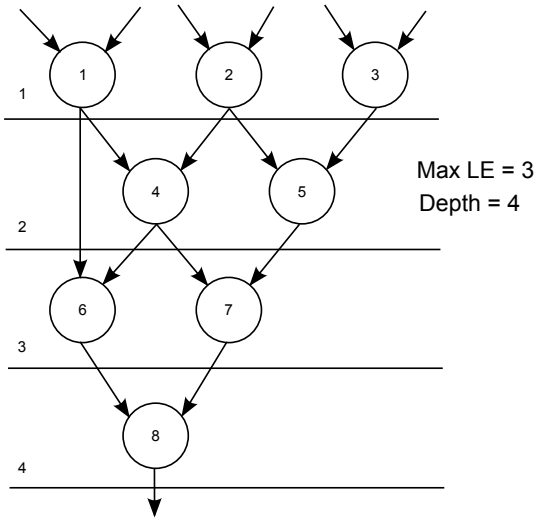


Figure 7.1: Without depth relaxation.

The performance of the proposed sub-optimal depth relaxation algorithm is demonstrated using a design example that has 8 LUT nodes scheduled for logic folding level-1 without depth relaxation is shown in the Figure 7.1, It can be observed that the depth of the circuit is determined to be 4. Across all the folding stages, the maximum number of LUTs required is 3. Hence by using 3 LUTs, the entire circuit can be realized to achieve optimum performance (minimum delay). By mapping the same circuit using sub-optimal depth relaxation method, the LUT usage can be minimized with slight increase in delay of the circuit. As shown in the Figure 7.2, by relaxing the depth of circuit from 4 to 6 by two steps, maximum number of LUT usage is reduced from 3 to 2. Similarly the LUT usage is reduced by 1 by relaxing the depth of the same circuit by 8. As the depth of the circuit is increased, the number of folding stages increases by an equal amount resulting in increased flexibility for scheduling the logic resources into more folding stages. This in turn reduces the number of LUTs in the folding stages, resulting in a higher probability of resource sharing without timing conflict.

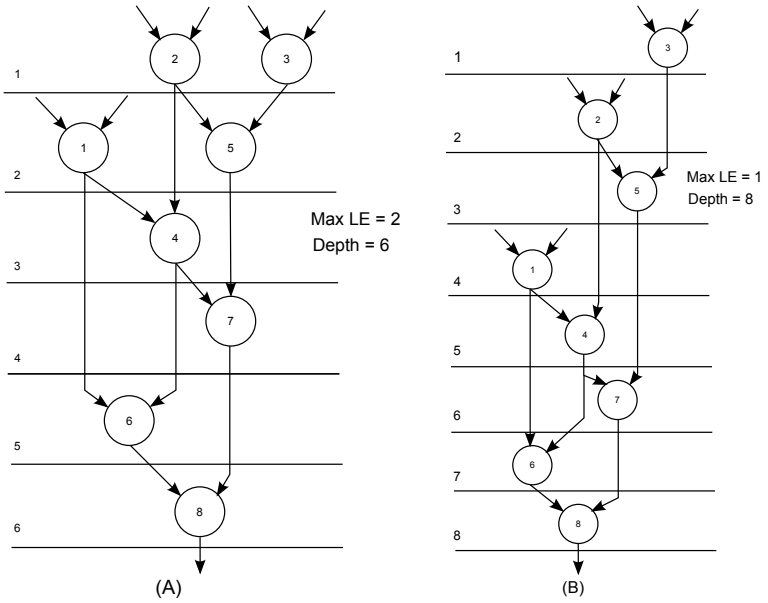


Figure 7.2: (A) Depth relaxed by 2 (B) Depth relaxed by 4.

7.3.3 Mapping Tool-Flow for Minimum P-D Evaluation

Current design tool flows require the designer to manually run the tool with all permutations of folding levels and DSP configuration settings in order to identify the minimum energy configurations. This section proposes an automated mapping tool that will automatically identify the minimum energy configuration for a given input design and user constraint. NATURE architecture supports mapping of four DSP configurations in all four folding levels (1, 2, 4 and 0). The four DSP configurations on each folding levels are:

1. Full-block reconfigurable DSP with fracturable multiplier configuration ($DSP_{config1}$)
2. Full-block reconfigurable DSP without fracturable multiplier configuration ($DSP_{config2}$)
3. Pipeline reconfigurable DSP block with fracturable multiplier configuration ($DSP_{config3}$)
4. Pipeline reconfigurable DSP block without fracturable multiplier configuration ($DSP_{config4}$)

With design space exploration being a critical parameter for dynamically reconfigurable architectures, the proposed automation tool incorporates different optimization schemes to identify the optimal and sub-optimal P-D product configuration based on user requirement in minimum iterations. The following sections describe different methods employed and its benefits.

(i) Exhaustive Search Technique

Exhaustive search is an iterative problem-solving method that consists of systematic enumeration of all possible choices for the solution and checks whether each choice satisfies the problem's statement. Exhaustive search techniques can be viewed as the simplest meta heuristic approach to find the solution when the problem size is limited. The exhaustive search technique is used as a baseline method for comparing the results generated from other proposed algorithms. As shown in Figure 7.3, the iterative process of the design automation tool starts by setting the folding level to 1 and DSP configuration parameter to $DSP_{config1}$ and evaluate corresponding area, resource usage, power and critical delay. After the first iteration, DSP configuration is changed to iterate through other 3 configuration sets such as $DSP_{config2}$, $DSP_{config3}$ and $DSP_{config4}$ respectively for folding level 1.

After the evaluation of all the four configuration sets of DSP block for level 1 folding, folding level is changed to 2 and the process is iterated. This method is repeated for folding level 4 and 0 respectively to generate a search space with P-D product value corresponding to all 16 configurations. For input designs where operating frequency is a user objective, the minimum P-D product corresponding to the constraint is sorted out from the generated search space. Similarly the minimum P-D product is sorted from the search space when the given input constraint is area.

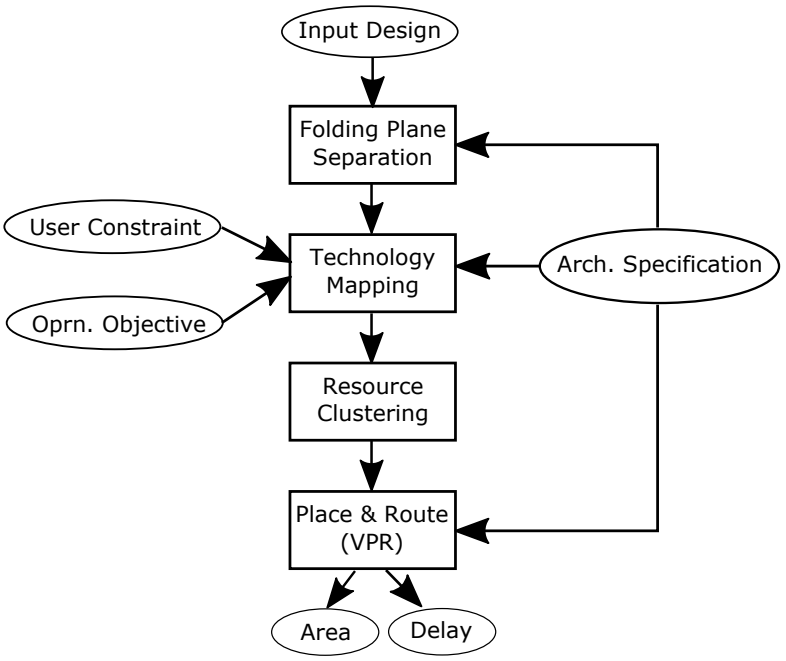


Figure 7.3: Existing FPGA architecture evaluation flow.

(ii) Cost Function Approach

Fig.7.3 illustrates the design automation tool flow considering area, delay and power [220] for NATURE architecture. For a given input design, the initial step is to optimize the logic followed by technology mapping in which the circuit is mapped onto basic building blocks of given architecture. Logic clustering is used to pack the mapped circuit to a given cluster size. After clustering, the VPR tool performs placement and routing to generate the final bitstream. Using the area and delay information obtained from VPR, the total power utilized by the circuit is estimated based on the switching activity of the components. Finally the P-D value corresponding to the given constraint is determined.

The design evaluation tool flow using exhaustive search techniques is a time consuming process. 80% of the total time consumed is taken up by VPR for an optimized placement and routing. This limits the faster evaluation of configuration corresponding to minimum P-D product for the given user constraints.

In cost function approach, the delay of the circuit is estimated based on the component connectivity of each folding stage using the NanoMap tool. The delay of basic building blocks (LUT, DSP, BRAM, and DFF) of NATURE architecture is

pre-calculated using Synopsys Design Compiler tool on 65 nm library to determine the circuit delay. Forward and backward edge tracing algorithms are incorporated in NanoMap to estimate the stage wise register to register combinatorial logic delay. If there exist a number of combinational logic within any folding stage, the maximum delay across the combinational logic of one folding stage is calculated. Once all the stage wise delays are estimated, the critical logic delay of the circuit is determined based on the maximum delay across all folding stages.

Next, the total area of the circuit is determined based on the individual area of each component using Synopsys Design Compiler on a 65 nm library. The automated tool iterates for 16 configurations and determines the area and power. At the end of the evaluation of area and delay for the entire configuration set, a cost function is evaluated for the minimum P-D product determination. The cost function is defined as:

$$CF = \alpha \times A_{norm} + \beta \times D_{norm} \tag{7.2}$$

where A_{norm} and D_{norm} are normalized area and delay respectively corresponding to each configuration and CF is the cost function. α and β are weight factors with:

$$\alpha + \beta = 1 \tag{7.3}$$

For the input design where only area is given as a user constraint compared to delay, the value of α is set to one and for applications where operating frequency is critical than area β can be set to one. The power consumption is only consumed by the used FPGA resources. The power model distinguishes different types of used hardware resources is as follows:

$$P_{total} = \sum_{i} \frac{1}{2} N_i \times f \times V^2 \times C_i \tag{7.4}$$

The summation is over different types of architecture elements, i.e., LUTs, DFF, DSPs and BRAMs. For type i circuit elements, C_i is the switching capacitance,

N_i is the number of used circuit elements, V is the voltage and f is the operating frequency of the resource. After determining total power of circuit, weighted P-D product is evaluated and is given by:

$$PD_{final} = P_{total} \times D_{ckt} \times CF \tag{7.5}$$

where PD_{final} is the final P-D product value, P_{total} is the total power evaluated from 7.4, D_{ckt} is the critical delay of the circuit and CF is the cost function from 7.2. Once PD_{final} is determined for all the 16 configurations, the minimum PD_{final} is sorted out. Finally, the design automation tool generates the configuration bits with configuration settings corresponding to identified minimum PD_{final}.

In the cost function approach, the delay parameter to estimate the minimum P-D product depends only on the component delay determined in the NanoMap tool flow. Hence, a significant reduction in compilation time can be achieved as the delay is not evaluated using VPR tool which consumes a considerable amount of time. Using this approach, it is possible to find out sub-optimal folding level and DSP configurations which might not result in the most efficient energy implementation. This is because routing delay has not been considered to estimate the circuit delay in this approach.

In the next section we discuss a model based technique which gives better P-D value estimation compared to the approach presented.

(iii) Multiple Linear Regression Technique

In the NATURE architecture, the P-D product estimation depends on folding level, DSP, LUT, and BRAM resource usage and the critical delay of the circuit. A multiple linear regression (MLR) method is used to model these dependencies, thereby reducing the need to run all the configurations to obtain the corresponding value of P-D.

MLR is one of the most widely used statistical techniques, that models relationship between a dependent variable and a set of independent variables/predictors [221]. Minimum energy profiling is performed using this linear fitting strategy over multiple regressor variables derived from design automation tools. The MLR technique basically consists of two steps:

- Generate a regressor model to estimate a dependent variable, Y based on certain defined independent/predictor variables, X. The estimation uses a subset of data samples (training samples) to generate a model.
- The regressor model is then used to estimate the dependent variable using independent variables corresponding to remaining data samples (test samples).

The MLR model estimation is given by:

$$Y_{est} = BX \tag{7.6}$$

where Y_{est} is the estimated value, B is the regression coefficients and X are the independent variables. Using least squares method, the MLR coefficients B can be estimated by

$$B = (X^T X)^{-1} X^T Y \tag{7.7}$$

The error of estimation is given by

$$E = Y - Y_{est} \tag{7.8}$$

and the Mean Square Error (MSE) for N number of data samples is given by:

$$MSE = \frac{1}{N} E E^T \tag{7.9}$$

Here, MLR is used to estimate the PD value (Y) corresponding to each configuration, using folding levels, total number of DSPs, SMBs and BRAM usage as the independent predictor variables (X).

First step in the MLR method is to create a model using randomly selected configuration pairs (folding level and DSP configuration) as training samples from the total configuration pair supported by NATURE. If the total number of configuration pair is *N*, the data *(Y,X)* corresponding to k configurations are randomly selected, where $1 < k < N$. The MLR model is generated based on these k configuration pairs. The model is then used to estimate Y for the remaining N-k configuration pairs and the MSE of the estimation is determined. For the same value of k, the process is repeated 100 times, and average value of MSE is considered, to avoid estimator bias. At the end of this method, average MSE corresponding to varying number of training samples (k $\in$ [1,N-1]) is determined. The value of k, that limits MSE within a threshold value is then determined, based on which the final model is created.

The total compilation time, i.e., the time required to obtain P-D values for each configuration is given by

$$T_{PD} = T_{VPR} + T_{NM} \tag{7.10}$$

where, T_{PD} is the total compilation time, T_{VPR} and T_{NM} represent the time required to run VPR and NanoMap tool respectively. Also, T_{VPR} accounts for roughly 80% of the total compilation time. The time required to determine configuration offering minimum P-D for a particular benchmark, in exhaustive search method is $N \times T_{PD}$. Using the MLR approach of estimating P-D, this time is reduced to $K \times T_{VPR} + N \times T_{NM}$, with a tolerable error in estimation. Through experiments, the optimum folding level and DSP configuration can be identified in the minimum iterations within a tolerable error range of 18%.

This approach achieves the minimum P-D configuration with better accuracy over cost function techniques for the multi-context NATURE platform that supports 16 configurations. Typically, MLR techniques provide accurate estimation for

large search space problems. Based on the regression model that is generated to estimate the optimal value, identical results can also be achieved for multi-context platforms that support large configuration sets.

However, the quality (in terms of accuracy) of estimation is affected as the number of predictor variables and data samples (total number of configurations) reduces resulting in poor regressor model generation. Hence, this technique is not suitable for finding the optimum solution for problems that have small search space due to the boundary constraints.

The next section introduces an approach for fast boundary identification of data samples (search space) based on the user constraints. This technique can be further extended to find out the sub-optimal and global optimal P-D product configurations (folding level, DSP configuration).

(iv) Nearest Neighbour Search for fast Boundary identification

It is often required to find the configuration corresponding to minimum energy for applications like FIR filter designs where operating frequency is a critical parameter. Moreover, it is equally important to realize such applications with emphasis on throughput over area, requiring the designer to implement the designs without going beyond a given A-D product. Under such scenarios, it is required to identify the upper and lower cut-off frequency and find the energy efficient configuration at the minimum number of iterations. To achieve this, a Nearest Neighbour Search (NNS) approach is used to explore and determine the upper and lower delay constraint in minimum iterations.

NNS is an optimization problem for finding the closest point or the boundary. NNS can be defined as follows: given a set of points P in a space R and a query $X \in R$, find the closest point in P to X. To find the upper delay constraint, the evaluation process (using NanoMap including VPR) starts with the configuration

set corresponding to maximum performance; i.e., folding level and DSP configuration set to zero and $DSP_{config4}$ and the ratio of resultant delay and the user constraint is determined. The ratio of delays is a measure of how close is the resultant delay to the user constraint delay. During the next iteration, only the folding level is changed from 0 to 4 and the tool evaluates area and delay for the new configuration. If the new ratio is less than the previous evaluation, then the current delay value is closer to the constraint, else the delay obtained is moving further away from the desired boundary. In this process, instead of performing an exhaustive search, the proposed algorithm jump between two configuration sets close to the boundary until the desired configuration is obtained.

In order to determine the configuration for upper delay constraint, the process is repeated except that the starting configuration set will be equal to the maximum delay point which is folding level 1 and DSP configuration $DSP_{config1}$. Once the closeness of delay is determined with respect to the given upper delay constraint, the automation tool (NanoMap including VPR) will evaluate the delay corresponding to folding level 2 and DSP configuration $DSP_{config1}$. Once the closest folding level is evaluated, the DSP configuration is changed and the ratio is further calculated. The iteration is carried out till the delay nearest to the upper delay constraint is obtained and the equivalent configuration is saved. It will be shown later that, by incorporating a NNS approach, the number of iterations to achieve folding level and DSP configuration pair corresponding to upper and lower delay constraints is reduced considerably. Moreover, if the upper and lower delay constraint values are closer to the extreme boundaries of the search space, then the configurations can be determined through minimum iterations.

Once the configuration pair corresponding to upper and lower delay constraints are identified, sub-optimal or global optimal P-D value configuration can be estimated using hill descent or heuristic approach respectively.

(iv-a) Hill descent Technique

The implementation of best energy efficient approach can be compromised to a certain degree for applications where time is a very critical parameter. For such cases, mapping of input circuit design with the configuration settings (folding level and DSP configuration) that satisfy within the user constraint is the best possible solution. Sub-optimal optimization algorithm like the greedy technique/hill descent [222] is introduced that makes a locally optimal choice for determining configuration pairs within a given user constraints. A greedy algorithm does not in general deliver an optimal solution, but nonetheless it may yield locally optimal solutions that approximate a global optimal solution within a *reasonable time*.

The proposed method starts from the folding level and DSP configuration value corresponding to the upper delay constraint obtained from the NNS approach and maps the circuit with the subsequent configuration to determine the P-D product. The delay value is determined after placement and routing using the VPR tool. During the first iteration, the resultant P-D value is compared against the value evaluated using the configuration pair corresponding to the upper delay constraint. If the resultant P-D value is less than the estimated P-D corresponding to the upper delay constraint, the configuration value is changed and the process is iterated until the resultant P-D value is higher than the previous P-D value. Finally, the design automation tool maps the input circuit with the folding level and DSP configuration corresponding to the lowest P-D value within the performed iterations.

For example, if the folding level and DSP configuration corresponding to the upper delay constraint is 2 and $DSP_{config3}$, the tool will map the application with the next configuration pair, folding level 2 and DSP configuration $DSP_{config4}$ and evaluates the P-D product. If the P-D product determined is greater than the previous P-D value, the previous configuration is determined as the locally optimal solution for minimum PD value within the constraint. However, if the current P-D product is less than the previous value, then the configuration is changed to

folding level 4 and $DSP_{config1}$ and the algorithm is iterated until the condition (current P-D value less than previous P-D value) violates.

The main benefit of the greedy approach is that the automation tool yields locally optimal energy efficient configuration parameters in a reasonable compilation time. The results prove that the proposed method is efficient for time critical applications.

(iv-b) Global Optimal

In this section, a heuristic method is proposed to select folding level and DSP configuration parameters corresponding to globally minimum P-D product within the user constraint delays. After determining the folding level and DSP configuration pairs corresponding to upper and lower delay user constraints using the NNS technique, the proposed design automation tool evaluates the P-D product delay for the configuration parameters that follow the parameter pair corresponding to upper delay constraint. This process is repeated until the configuration pair corresponding to the lower delay constraint is reached. Once all the P-D products are evaluated, folding level and DSP configuration corresponding to the minimum energy are determined.

The results show that the total time take to identify the minimum energy configuration set is greater than the locally optimal method, since the tool runs through all configuration pairs between the upper and lower delay constraints. However, the proposed global optimal method consumes relatively less computation time compared to an exhaustive search approach.

7.4 Results and Discussion

Experiments are performed using the same benchmark suites used in the previous chapter. Among 12 benchmarks, ARF, FIR1, FIR2 and EWF are popular benchmarks from MediaBench [193], having only directed acyclic datapaths, which

mainly contain multiplications, additions and subtractions. Additionally, ASPP4 (Application Specific Programmable processor), DCT (discrete cosine wave transform), Diffeq, Paulin (differential-equation solvers), and Biquad (digital filter) contain both datapaths and control logic with cyclic dependencies. Complex benchmarks such as HornerBezier, MotionVector, MatrixMult, and SmoothTriangle are dominated by arithmetic and memory operations.

7.4.1 Performance Evaluation of Depth Relaxation

This section presents the experimental results for several benchmarks mapped onto the fine-grained NATURE architecture using modified NanoMap tool for sub-optimized depth relaxation algorithm. For all benchmarks used in the experiment, logic circuits are folded using folding level-1. The experiment is carried out in three different modes of mapping tool:

- Mapping with depth relaxation without delay constraints.
- Mapping with depth relaxation with delay constraints.
- Mapping without depth relaxation.

The depth relaxation optimization in NanoMap can map the benchmark suites to its optimal area or to its optimal performance. The proposed algorithm is flexible enough to map the benchmarks without compromising the performance by relaxing the depth with delay constraint. The experiment is performed for the benchmarks on all the three modes and the result are shown in the Table 7.1. From the table, it can be observed that for GCD benchmark under a given delay constraints when the depth is relaxed by 2 steps (from folding stage 21 to 23), the LE resource usage is reduced from 32 to 17, resulting in 46.8% LE reduction. For the same benchmark without delay constraint, similar reduction in LE resource usage is observed. However in ARF benchmark, it can be observed that the there is no considerable reduction in LE usage even when the depth is relaxed. This is because even after relaxing the depth, certain scheduled nodes along the folding

Benchmarks	Without depth relaxation		With depth relaxation			
			Without delay		With delay	
	LE_count	Depth	LE_count	Depth	LE_count	Depth
Paulin	84	34	80	36	80	36
EX2	50	26	50	26	47	29
GCD	32	21	17	23	17	23
ASPP4	100	29	100	29	96	32
Biquad	76	28	68	30	64	32
Cos1	282	25	282	25	272	30
arf	504	25	495	28	495	28

Table 7.1: Comparison of LE usage with and without depth relaxation.

stage remain unchanged. Hence the number of LE resources required to realize the logic in that stage remains the same irrespective of depth. An average LE reduction of 11.8% and 9.13% is achieved for depth relaxation with and without delay constraints over mapping of circuits without depth relaxation.

7.4.2 Performance Evaluation of Energy Efficient Tool-Chain

The performance evaluation (speedup and % error) of the proposed optimization techniques that determine optimal configuration for minimum P-D product is presented in this section. The exhaustive search technique is performed to be used as a ground truth to identify the minimum energy configuration corresponding to user defined constraint for all the 16 configurations. The average compilation time (in seconds) for each benchmarks over 100 iterations for minimum energy configuration is determined. These results are further used to evaluate the benefits of various technique proposed in this chapter.

(i) Comparison between Exhaustive and Cost Function Technique

In the cost function approach, based on the user constraints (area or delay), the α and β values are evaluated and determine the minimum P-D configuration.

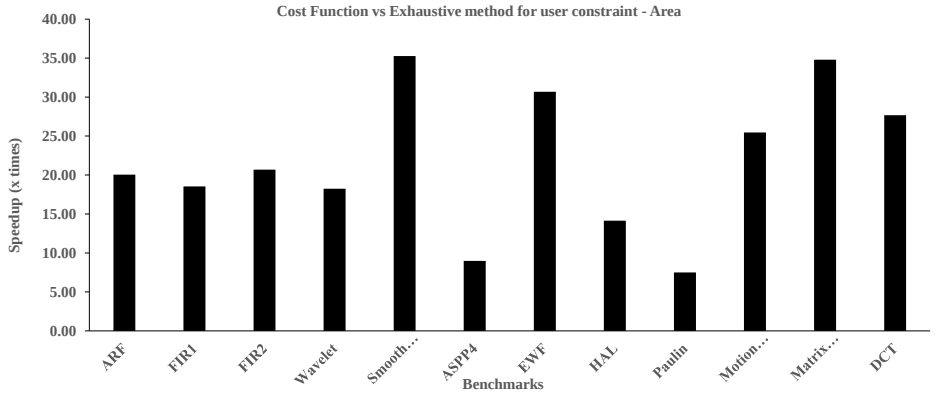


Figure 7.4: P-D product Speedup for Cost Function Vs Exhaustive Search.

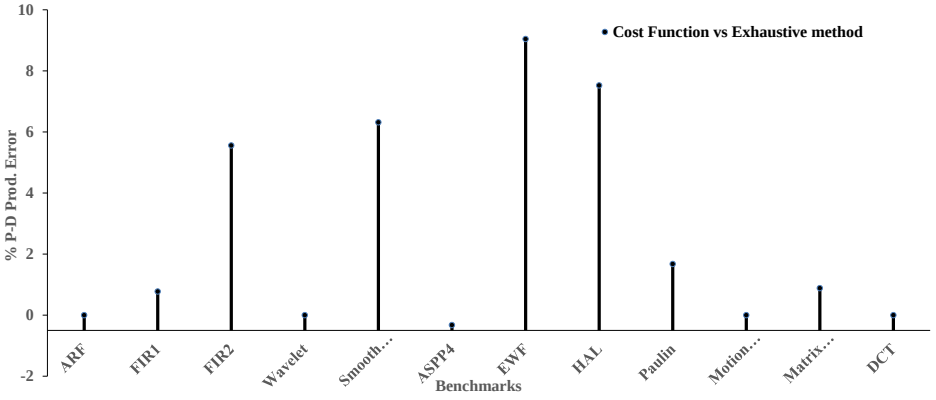


Figure 7.5: P-D product % Error for Cost Function Vs Exhaustive Search.

The compilation time taken by the proposed method is also evaluated. The obtained results are then compared against the baseline approach (exhaustive search method) for the same user defined constraints. It can be observed from Figure 7.4 that the proposed cost function techniques achieves significant compilation time speedup of 21.83× over an exhaustive search method. Speedup is achieved mainly by estimating the critical delay of the circuit using the proposed component delay estimation method employed in the NanoMap. This avoids the compilation time overhead caused by the VPR tool during placement and routing. Although the routing delay is not considered during P-D product estimation, the experimental results show that the deviation of estimated P-D value from the original value is in tolerable range from -0.003 to +0.09. The P-D product % error for each benchmark is shown in Figure 7.5.

Table 7.2: Comparison between Exhaustive and MLR Method.

Benchmark	Exhaustive Search			MLR Technique			% Error	Speedup
	Original PD	Comp. Time (s)	Total Run Count	Est. PD	Comp. Time (s)	Training Samples		
ARF	2.11	809.18	16	2.25	380.52	8	6.64	2.13×
FIR1	1.29	509.21	16	1.29	274.43	8	0.00	1.86×
FIR2	1.26	599.30	16	1.34	356.11	9	6.35	1.68×
Wavelet	4.16	801.1	16	4.16	401.70	8	0.00	1.99×
Smooth Triangle	5.70	1204.55	16	5.79	700.89	9	1.58	1.72×
ASPP4	3.09	362.50	16	3.09	287.62	9	0.00	1.26×
EWF	1.99	1098.2	16	1.99	677.30	9	0.00	1.62×
HAL	9.97	402.50	16	9.97	221.47	8	0.00	1.82×
Paulin	3.58	261.74	16	3.64	253.10	9	1.68	1.03×
DCT	2.82	863.70	16	2.89	512.23	8	2.48	1.69×
Motion Vector	2.34	768.30	16	2.34	367.48	7	0.00	2.09×
MatrixMult	1.13	2054.8	16	1.13	1304.80	9	0.00	1.57×

(ii) Comparison between Exhaustive and MLR Technique

The accuracy of the MLR technique is determined by how well the regressor model is generated based on the training samples and predictor variables. The challenge here is to determine the minimum number of training samples required to model the MLR, so as to estimate the P-D for all configurations with minimum MSE. The training samples are randomly selected from the total (16 configurations) samples. The process is repeated N (100) times, using random training samples each time, to create the model. The average performance of all these models is considered to avoid bias (or to generalize) in results. The minimum number of training samples required to limit MSE within a threshold value is then determined, based on which the model is created. A threshold of 18% of the maximum MSE is chosen (empirically). The estimated P-D value and computation time of the MLR technique is then compared against the baseline approach, as shown in the Table 7.2. It can be observed that out of the 12 benchmarks, P-D value obtained for seven benchmarks using proposed MLR method matches the desired P-D value, resulting in 0% error. The results show an efficient P-D estimation with minimal MSE (1.56%) can be achieved using only 9 training samples. For benchmark like Motion Vector, it requires only 7 training samples i.e., running both NanoMap and VPR for 7 configurations, to generate the MLR model to estimate the P-D value, achieving a speedup of 2.09×. The proposed MLR technique reduces the need to run the VPR tool from 16 (for exhaustive search) to 9 times, offering a 1.71 × speedup on average.

Table 7.3: Comparison between Exhaustive and Hill Descent Method.

Benchmark	Exhaustive Search			Sub-Optimal			% Error	Speedup
	Original PD	Comp. Time (s)	Total Run Count	Est. PD	Comp. Time (s)	Total Run Count		
ARF	4.27	809.18	16	4.27	361.82	7	0.00	2.24×
FIR1	1.29	509.21	16	1.29	287.75	8	0.00	1.77×
FIR2	4.01	599.3	16	4.16	303.40	7	3.74	1.98×
Wavelet	1.12	801.1	16	1.23	325.01	7	9.82	2.46×
Smooth Triangle	5.70	1204.64	16	6.07	488.68	7	6.49	2.47×
ASPP4	8.12	362.5	16	8.21	192.80	7	1.11	1.88×
EWF	1.99	1098.2	16	1.99	552.30	7	0.00	1.99×
HAL	1.08	402.5	16	1.09	200.73	7	0.93	2.01×
Paulin	6.69	261.74	16	7.35	116.10	7	9.87	2.25×
DCT	2.82	863.7	16	2.89	417.60	7	2.48	2.07×
Motion Vector	4.81	768.3	16	4.81	607.36	10	0.00	1.26×
MatrixMult	1.15	2054.8	16	1.37	1072.30	7	19.13	1.92×

(iii) Comparison between Exhaustive, Sub-optimal and Globally Optimal Techniques

This section quantifies the performance matrix (speedup, % error and total configuration run) achieved for the proposed sub-optimal and global optimal techniques for minimum P-D product configuration for bounded delay constraints. Based on the user constraint given (lower cut-off frequency) and maximum A-D product for each benchmark, the upper and lower delay constraint using NNS technique is determined. Compared to an exhaustive approach, the NNS method improves the design productivity by identifying the upper and lower configuration parameters (folding level and DSP configuration parameters corresponding to user defined upper and lower delay constraints is defined as upper and lower configuration parameters) in a minimum number of iterations. After the bounded delay estimation, the proposed integrated tool flow executes with the designer choice for sub-optimal or global optimal value for minimum energy configuration identification. The sub-optimal P-D product configuration is determined using a hill descent approach, while an exhaustive search mentioned is carried out to determine the global minimum P-D product between the bounded region.

For a fair comparison, the results obtained from the sub-optimal and global methods are compared against a baseline approach (exhaustive method) to illustrate the % error and speedup achieved. Table 7.3 illustrates comparison between sub-optimal and exhaustive method. It can be observed that an average speedup of 2.02× is achieved for the proposed sub-optimal method over exhaustive method

Table 7.4: Comparison between Exhaustive and Global Method.

Benchmark	Exhaustive Search			% Globally Optimal			% Error	Speedup
	Original PD	Comp. Time (s)	Total Run Count	Est. PD	Comp. Time (s)	Total Run Count		
ARF	4.27	809.18	16	4.27	671.94	13	0	1.20×
FIR1	1.29	509.21	16	1.29	443.10	15	0	1.15×
FIR2	4.01	599.3	16	4.01	498.94	13	0	1.20×
Wavelet	1.12	801.1	16	1.12	566.70	13	0	1.41×
Smooth Triangle	5.70	1204.64	16	5.70	1153.30	14	0	1.04×
ASPP4	8.12	362.5	16	8.12	321.40	13	0	1.13×
EWF	1.99	1098.2	16	1.99	998.00	13	0	1.10×
HAL	1.08	402.5	16	1.08	331.81	13	0	1.21×
Paulin	6.69	261.74	16	6.69	192.73	13	0	1.36×
DCT	2.82	863.7	16	2.82	837.60	14	0	1.03×
Motion Vector	4.81	768.3	16	4.81	668.57	15	0	1.15×
MatrixMult	1.15	2054.8	16	1.15	1742.00	13	0	1.18×

at minimum configuration iteration. For benchmarks like ARF, FIR1, EWF, and Motion vector, the sub-optimal technique provides accurate configuration parameters for minimum P-D product value.

However, the sub-optimal technique is error prone when compared to the exhaustive (brute-force) technique. It can be observed from the Table 7.3 that the configuration parameters determined using hill descent method (sub-optimal) for 8 out of 12 benchmark suites is different from desired configuration parameters. This is because of the fact that the PD values determined using exhaustive search method in Table 7.2 correspond to the minimum PD product for each benchmark across all 16 configuration without user defined delay constraint, where as in tables 7.3 and 7.4, the PD values determined correspond to the minimum PD estimated for the bounded delay constraint given by the user for each benchmark. The average P-D product % error across all benchmarks is observed to be 4.46%.

Using the proposed global optimization technique for the bounded delay constraint, it can be observed that the P-D product value achieved (0% error) is the same as that of exhaustive search method. Table 7.4 shows the comparison between global and exhaustive methods. An average speedup of 1.18× is achieved for proposed method over the baseline approach. Furthermore, the configuration parameters determined using the global optimal method matches the desired configuration settings resulting in 0% error. This is achieved by running the NanoMap tool for more iterations. Compared to sub-optimal method, an average increase in total run count of 1.85× is required for global optimal method.

7.5 Summary

This chapter proposed an intelligent automated NanoMap tool that maps an input circuit efficiently by choosing the best configuration supported by NATURE architecture that meets the desired A-D/P-D product. To achieve the best A-D trade-off, a sub-optimal depth relaxation algorithm is proposed aiming at minimizing the area without considerably affecting the performance. Based on the user delay constraint, the tool will map the input circuit to a minimum area by relaxing depth. From the simulation results of various benchmarks, a maximum reduction of 15 LEs is observed in the GCD benchmark. Hence by incorporating the sub-optimized depth relaxation method in dynamic reconfigurable architecture tool flow, it is possible to implement different applications using minimum hardware resource.

Also, different optimization techniques are incorporated in the NanoMap tool to determine minimum P-D product and the corresponding configuration parameters in minimum runtime. The proposed cost function technique achieved a 21.83× improvement in compilation time over exhaustive search methods within an error tolerance range between -0.0003 to +0.09. Using a model based (MLR method) estimation to determine the configuration pair for minimum P-D value, compilation time speedup of 1.64× is achieved over exhaustive search method. For fast user constraint determination within the search space, nearest neighbor search method is incorporated. A sub-optimal and global optimal technique to estimate the minimum P-D product within a bounded user defined constraints is also proposed. Experimental results show that the proposed method achieved 2.02× and 1.18× improvement in compilation time over exhaustive search method. The optimization techniques incorporated in NanoMap give the flexibility for user to choose the best technique that suites the application requirement (minimum configuration run or better accuracy).

8

Conclusion and Future Research

Hybrid multi-context FPGA architectures like NATURE incorporate high-density, high-speed nano RAMs to enable runtime cycle-by-cycle reconfiguration and temporal logic folding. In this thesis, several unique DSP architectures are proposed for the hybrid multi-context NATURE platform in order to better exploit its advantages and possibilities. The dynamic reconfiguration of DSP blocks at full-block and pipeline levels provides designer flexibility to achieve A-D trade-offs while implementing different arithmetic operations. The reconfigurable multi-precision DSP block proposed in this thesis also enables the NATURE architecture to realize mixed-precision applications efficiently.

The research also investigated different design space exploration algorithms that can determine the optimal configuration for a given circuit based on the design requirements and user constraints. These are achieved by incorporating different

optimization algorithms such as cost function, MLR and NNS based techniques to the NanoMap tool flow to automatically explore the different folding levels and DSP modes. This chapter draws the conclusion from various contributions presented in this thesis and outlines future research direction.

8.1 Summary of Contributions

Three DSP architectures optimized for NATURE platform are proposed in this thesis. To support optimized mapping of these DSPs for the NATURE platform, intelligent algorithms are also developed and incorporated into the NATURE NanoMap based tool chain.

8.1.1 Full-Block Reconfigurable DSP Block

Chapter 4 of this thesis proposed a novel full-block reconfigurable DSP architecture that can be incorporated with the NATURE architecture to perform compute intensive arithmetic operations. The DSP block has 3 pipeline stages which contain dual pre-adders, 16×16 Wallace tree multiplier, and 32-bit ALU unit and can operate up to a maximum frequency of 300 MHz. The flexible interconnects and integration of multiple output registers enable it to bypass unused functions and store the results so that it can be reused to realize other functions. The NanoMap tool is correspondingly extended such that it can support the efficient mapping of the proposed DSP blocks for the NATURE platform. Simulation results based on 7 complex mathematical benchmarks demonstrated that the delay and LE usage on the proposed DSP incorporated architecture is less than the fine-grained NATURE architecture. By mapping the complex arithmetic operations along the critical path using DSP blocks and other random logic using LEs, the performance can be improved by 58.6% on average.

8.1.2 Pipeline Reconfigurable DSP Block

The limitation of the proposed full-block reconfigurable DSP is that it can only be reused to map arithmetic operations that are scheduled after one full DSP operation. Hence, the full-block DSP fails to fully exploit the capability of NATURE's temporal logic folding, resulting in under utilization of pipeline stages. Chapter 5 then presented a pipeline reconfigurable DSP architecture that supports independent reconfiguration of pipeline stages to enable efficient utilization of NATURE's logic folding feature. A 4-stage pipelined DSP block which uses a 16-bit BW multiplier with HPM reduction tree is used to achieve higher power efficiency and better performance when compared to Wallace tree multiplier. A clock gating technique is incorporated in individual pipeline stages to minimize the power consumption of unused stages. The NanoMap tool is also further extended to support the proposed DSP blocks, such as to automate the mapping of complex arithmetic operations using the DSP blocks in NATURE circuit realization. Experimental results demonstrated that the pipeline reconfigurable DSP block is able to fully exploit the temporal logic folding capability of NATURE, thereby minimizing the area overhead and increasing the effective logic density utilization. The total power consumption of circuits realized can also be minimized by clock gating the individual pipeline stages of the proposed DSP blocks. Compared to full-block DSPs based design, the efficient reuse of reconfigurable DSP blocks enable an average of 31.42% reduction in area and 4.18× improvement in P-D product across different benchmarks. A comparison between the proposed DSP incorporated NATURE architecture with folding level-0 and Spartan-3A DSP FPGA is also performed. The results show that an average improvement in performance of 1.29× and 54.13% gain in area can be achieved.

8.1.3 Fracturable DSP Block

While the two DSP blocks described earlier can be used to achieve efficient resource and power reduction in a given circuit, for RTL designs with mixed precision and

custom datapath widths, usage of fixed precision DSP blocks often result in sub-optimal utilization. To overcome these problem, a fracturable DSP architecture was presented in chapter 6 that allows its internal compute-path to be fractured while maintaining the capability to dynamically switch computation precision. By utilizing this capability, the proposed DSP block can switch between sub-width operation mode (2 independent 8×8 operations simultaneously), full width operation mode, or wider multiplication mode (32×32, 24×16 and 24×8 on a single DSP) at runtime. The fracturable DSP block can efficiently handle two independent half-width (8×8) multiplications in complete isolation, perform a single 8×8 multiplication with lower power consumption or operate on regular full-width 16-bit operands. The NanoMap tool flow is further extended to efficiently map and merge mixed precision multiplications on the proposed DSP block. Experimental results based on mapping benchmarks circuits onto NATURE showed that architecture that incorporates the proposed DSP block can further achieve 42.5% and 53.7% average reduction in area and DSP block utilization with 2.1× improvement in energy efficiency without utilizing temporal folding (folding level 0). The proposed method also demonstrated an improvement in energy efficiency and resource utilization when temporal folding is employed, without sacrificing performance.

8.1.4 Area/Power-Aware NanoMap

Optimal mapping of input circuits on a multi-context FPGA platform requires an equally efficient mapping tool for converting the given circuit to the configuration bitstream. Chapter 7 proposed an intelligent automated NanoMap tool that can determine the best configuration supported by NATURE architecture to meet the desired A-D/P-D product. A sub-optimal depth relaxation algorithm was used to minimize the area without affecting the performance excessively. Based on the user delay constraint, the tool will map the input circuit to a minimum area by relaxing its depth.

Various optimization algorithms are incorporated in the NanoMap tool that can determine the minimum P-D product and the corresponding configuration parameters in a minimum runtime. First, a cost function based technique was proposed to determine the minimum energy configuration in which the critical delay of the circuit is estimated using component delay estimation incorporated in NanoMap. The proposed cost function technique achieved 21.83× improvement in compilation time over exhaustive search methods within an error tolerance range between -0.0003 to +0.09. Cost function approach can be used for circuits that need to be mapped on the fly with configuration for minimum P-D product. Using a model based (MLR method) estimation to determine the optimum folding level and DSP mode for minimum P-D value, the compilation time improvement of 1.64× was achieved over an exhaustive search method. This technique is suitable for a multi-context FPGA architecture that supports a wide range of configuration. A sub-optimal and global optimal technique to estimate the minimum P-D product within bounded user defined constraints was also proposed. For fast user constraint determination within the search space, a nearest neighbour search method was incorporated. Experimental results show that the proposed method achieved 2.02× and 1.18× improvement in compilation time over an exhaustive search method. NNS-based sub-optimal and global optimal techniques provide solution for P-D estimation with user constraints. Hence, incorporating the proposed algorithms in NanoMap allows the designer to choose the required approach that meets the user constraint.

8.2 Future Research

The following are several possible extensions to the work presented in this thesis, which can be explored in future research.

8.2.1 Floating Point DSP block for Complex Arithmetic Computations

The DSP blocks presented in this thesis only support fixed-point arithmetic computations. The growing complexity of DSP algorithms, dynamic range of data representations and its precision demand the need of DSP block that can perform floating point computations. Moreover, mapping and computing real world critical applications in the field of space research, medical imaging etc. require the DSP block to deal with floating point data with high accuracy. Mapping such applications on current NATURE architecture require cascading of multiple DSP blocks which is highly inefficient and power consuming. As such, the existing NATURE architecture can be extended to 32-bit with floating DSP block to support dynamic range of data representation.

8.2.2 Efficient Template Matching of DSP Block

Currently in the technology mapping phase, each arithmetic node is independently mapped onto an individual DSP block. After the scheduling phase, the non-overlapping DSP operations are clustered to one physical DSP block, thereby reducing the total resource utilization. However, for folding level-0 (no logic folding) the clustering of DSP operations to one physical DSP block does not takes place and results in large resource utilization. Moreover, the full advantage of the proposed DSP block with its sub-blocks that provide different functionality is not fully exploited. Template matching techniques could be explored in order to fully exploit the wide range of configuration of dual pre-adder based DSP block for folding level-0. In this technique, after technology mapping, the DFG is segmented into sub-graphs that match various possible configurations of the DSP block primitive. This would provide full utilization of DSP and its sub-blocks for folding level-0, resulting in minimum resource usage of DSP blocks thereby minimum area and power consumption.

8.2.3 Improved Temporal Clustering Algorithm

As opposed to the conventional clustering methods, clustering in NanoMap supports temporal logic folding techniques. Clustering aims to reduce the resource utilization by taking the flattened LUT and DSP network as input and group the non-overlapping LEs and DSPs into physical LEs and DSP blocks respectively. The current clustering strategy incorporates a constructive algorithm that tries to reduce the number of LUT usage by packing them as much as possible into the SMBs, is thus, primarily focused on minimizing area. However, this can have adverse effects on delay and power. Hence, it is required to have a clustering scheme that achieves a balance between both delay and area to meet the user constraint for better A-D trade-offs. This can be done by incorporating improved clustering techniques like simulated annealing to devise a more balanced approach.

8.2.4 High-Level Synthesis (HLS) Tool for NATURE

Designing complex systems using HDL requires expertise to efficiently describe the circuit behaviour and to optimize the control/data flow patterns for high performance. HLS tools aim to lower this barrier, allowing the system behaviour to be described at a higher level of abstraction using standard C/C++ programming languages. However, in NATURE architecture, the input design has to be described in the form of a low-level technology mapped netlist, called ($.mg$) format. It requires the designer to have a fair understanding of the NATURE architecture and its capabilities like temporal folding, limiting the adoption of NATURE for complex systems. Also, traditional HLS algorithms perform allocation, scheduling and binding to generate the RTL description, which prevents effective use of temporal folding in multi-context architectures. To overcome these limitations, a dedicated HLS tool-flow for NATURE can be developed. The tool will read the circuit described in C/C++ format and translate to the optimized netlist, which can be further used by the NanoMap tool flow to generate the configuration bitstream.

8.3 Summary

This thesis has contributed several DSP architectures and efficient mappings for multi-context NATURE architecture to implement compute intensive arithmetic kernels, enabling the designer to better exploit the capabilities for mapping complex applications. The results from this work will be beneficial to the FPGA community in migrating to next generation of FPGA devices such as the NATURE based architecture for a wide range of applications.

Bibliography

[1] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*, volume 497. Springer Science & Business Media, 2012.

[2] [Online] Stratix Architecture. `http://www.altera.com/literature/hb/stx/ch_2_vol_1.pdf`.

[3] Seth Copen Goldstein, Herman Schmit, Matthew Moe, Mihai Budiu, Srihari Cadambi, R Reed Taylor, and Ronald Laufer. PipeRench: A co/processor for Streaming Multimedia Acceleration. In *ACM SIGARCH Computer Architecture News*, volume 27, pages 28–39. IEEE Computer Society, 1999.

[4] [Online] Stratix Architecture. `https://www.altera.com/products/fpga/features/dsp/stratix-v-dsp-block.html`.

[5] Xilinx Inc. *UG369: Virtex-6 FPGA DSP48E1 Slice User Guide*, 2011.

[6] Hadi Parandeh-Afshar, Alessandro Cevrero, Panagiotis Athanasopoulos, Philip Brisk, Yusuf Leblebici, and Paolo Ienne. A Flexible DSP Block to Enhance FPGA Arithmetic Performance. In *Proceedings of International Conference on Field-Programmable Technology, 2009. FPT 2009.*, pages 70–77. IEEE, 2009.

[7] Yuan Lin, Hyunseok Lee, Mark Woh, Yoav Harel, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti, and Krisztian Flautner. SODA: A High-Performance DSP Architecture for Software-Defined Radio. *IEEE Transactions on Micro*, 27(1):114–123, 2007.

[8] Wei Zhang, Niraj K Jha, and Li Shang. A Hybrid Nano/CMOS Dynamically Reconfigurable System Part I: Architecture. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 5(4):16, 2009.

[9] Wei Zhang, Niraj K Jha, and Li Shang. Design Space Exploration and Data Memory Architecture Design for a hybrid Nano/CMOS Dynamically Reconfigurable Architecture. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 5(4):17, 2009.

[10] Wei Zhang, Niraj K Jha, and Li Shang. A hybrid Nano/CMOS Dynamically Reconfigurable System Part II: Design Optimization Flow. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 5(3):13, 2009.

[11] Katherine Compton and Scott Hauck. Reconfigurable computing: A Survey of Systems and Software. *ACM Computing Surveys (csuR)*, 34(2):171–210, 2002.

[12] [Online] Xilinx Product Specification. `http://www.xilinx.com/publications/matrix/Product_Selection_Guide.pdf/`.

[13] [Online] Altera Product Catclogue. `http://www.altera.com/literature/sg/product-catalog.pdf`.

[14] Andre DeHon. Dynamically Programmable Gate Arrays: A Step Toward Increased Computational Density. In *Proceedings of Fourth Canadian Workshop on Field Programmable Devices*, volume 47, 1996.

[15] Steven Trimberger, Dean Carberry, Anders Johnson, and Jennifer Wong. A Time-Multiplexed FPGA. In *5th IEEE International Symposium on FPGAs for Custom Computing Machines (FCCM), 1997.*, pages 22–28. IEEE, 1997.

[16] Stephen M Scalera and Jóse R Vázquez. The Design and Implementation of a Context Switching FPGA. In *Proceedings of IEEE International Symposium on FPGAs for Custom Computing Machines (FCCM), 1998.*, pages 78–85. IEEE, 1998.

[17] Kizheppatt Vipin, Suhaib Fahmy, et al. Efficient Region Allocation for Adaptive Partial Reconfiguration. In *Proceedings of International Conference on Field-Programmable Technology (FPT), 2011.*, pages 1–6. IEEE, 2011.

[18] Kizheppatt Vipin and Suhaib A Fahmy. Architecture-Aware Reconfiguration-Centric Floorplanning For Partial Reconfiguration. In *Reconfigurable Computing: Architectures, Tools and Applications*, pages 13–25. Springer, 2012.

[19] André DeHon, Michael J Wilson, and Charles M Lieber. Nanoscale Wire-based Sublithographic Programmable Logic Arrays, September 25 2007. US Patent 7,274,208.

[20] Greg Snider, Philip Kuekes, and R Stanley Williams. CMOS-like Logic in Defective, Nanoscale Crossbars. *Journal of Nanotechnology*, 15(8):881, 2004.

[21] Seth Copen Goldstein and Mihai Budiu. Nanofabrics: Spatial Computing Using Molecular Electronics. *ACM SIGARCH Computer Architecture News*, 29(2):178–191, 2001.

[22] Thomas Rueckes, Kyoungha Kim, Ernesto Joselevich, Greg Y Tseng, Chin-Li Cheung, and Charles M Lieber. Carbon Nanotube-Based Nonvolatile Random Access Memory for Molecular Computing. *science*, 289(5476):94–97, 2000.

[23] Stefan Lai. Current Status of the Phase Change Memory and its Future. In *Electron Devices Meeting, 2003. IEDM'03 Technical Digest. IEEE International*, pages 10–1. IEEE, 2003.

[24] Saied Tehrani, Jon M Slaughter, Mark Deherrera, Brad N Engel, Nicholas D Rizzo, John Salter, Mark Durlam, Renu W Dave, Jason Janesky, Brian

Butcher, et al. Magnetoresistive Random Access Memory Using Magnetic Tunnel Junctions. *Proceedings of the IEEE*, 91(5):703–714, 2003.

[25] Varghese George, Hui Zhang, and Jan Rabaey. The Design of a Low Energy FPGA. In *Proceedings of 1999 international symposium on Low power electronics and design*, pages 188–193. ACM, 1999.

[26] Deming Chen, Jason Cong, Yiping Fan, and Zhiru Zhang. High-level Power Estimation and Low-Power Design Space Exploration for FPGAs. In *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 529–534. IEEE Computer Society, 2007.

[27] Julien Lamoureux and Wayne Luk. An Overview of Low-Power Techniques for Field-Programmable Gate Arrays. In *Proceedings of International Conference on Adaptive Hardware and Systems, 2008. AHS'08. NASA/ESA.*, pages 338–345. IEEE, 2008.

[28] Jonathan Rose and Stephen Brown. Flexibility of Interconnection Structures for Field-Programmable Gate Arrays. *IEEE Journal of Solid-State Circuits*, 26(3):277–282, 1991.

[29] Alfred E Dunlop and Brian W Kernighan. A Procedure for Placement of Standard Cell VLSI Circuits. *IEEE Transactions on Computer-Aided Design (TCAD)*, 4(1):92–98, 1985.

[30] Dennis J-H Huang and Andrew B Kahng. Partitioning-Based Standard-Cell Global Placement With An Exact Objective. In *Proceedings of International Symposium on Physical design*, pages 18–25. ACM, 1997.

[31] JS Rose, WM Snelgrove, and ZG Vranesic. ALTOR: An Automatic Standard Cell Layout Program. In *Proceedings of Canadian Conference on VLSI*, pages 169–173, 1985.

[32] Jürgen M Kleinhans, Georg Sigl, Frank M Johannes, and Kurt J Antreich. GORDIAN: VLSI Placement by Quadratic Programming and Slicing

Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 10(3):356–365, 1991.

[33] Georg Sigl, Konrad Doll, and Frank M Johannes. Analytical placement: A Linear or a Quadratic Objective Function? In *Proceedings of 28th International ACM/IEEE Design Automation Conference (DAC)*, pages 427–432. ACM, 1991.

[34] Charles J Alpert, Tony F Chan, Andrew B Kahng, Igor L Markov, and Pep Mulet. Faster Minimization of Linear Wirelength for Global Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 17(1):3–13, 1998.

[35] Charles J Alpert, T Chan, DJ-H Huang, I Markov, and Kenneth Yan. Quadratic Placement Revisited. In *International Proceedings of the 34th annual Design Automation Conference (DAC)*, pages 752–757. ACM, 1997.

[36] Arvind Srinivasan, Kamal Chaudhary, and Ernest S Kuh. RITUAL: A Performance Driven Placement Algorithm for Small Cell ICs. In *Proceedings of IEEE International Conference on Computer-Aided Design, (ICAD) 1991.*, pages 48–51. IEEE, 1991.

[37] Scott Kirkpatrick. Optimization by Simulated Annealing: Quantitative Studies. *Journal of statistical physics*, 34(5-6):975–986, 1984.

[38] William Swartz and Carl Sechen. Timing Driven Placement for Large Standard Cell Circuits. In *Proceedings of 32nd International Symposium on Design Automation Conference, 1995. DAC'95.*, pages 211–215. IEEE, 1995.

[39] Carl Sechen and Kai-Win Lee. An Improved Simulated Annealing Algorithm for Row-Based Placement. In *Proceedings of IEEE International Conference on Computer Aided Design (CAD)*, pages 478–481, 1987.

[40] Carl Sechen and Alberto Sangiovanni-Vincentelli. The TimberWolf placement and routing package. *IEEE Journal of Solid-State Circuits*, 20(2):510–522, 1985.

[41] Carl Ebeling, Larry McMurchie, Scott A Hauck, and Steven Burns. Placement and routing tools for the Triptych FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 3(4):473–482, 1995.

[42] Sudip K Nag and Rob A Rutenbar. Performance-Driven Simultaneous Place and Route for Island-Style FPGAs. In *Proceedings of International Conference on Computer-Aided Design, 1995. ICCAD-95. Digest of Technical Papers.*, pages 332–338. IEEE, 1995.

[43] Michael J Alexander, James P Cohoon, Joseph L Ganley, and Gabriel Robins. An Architecture-Independent Approach to FPGA Routing Based on Multi-Weighted Graphs. In *Proceedings of International Conference on European design automation (EDA)*, pages 259–264. IEEE Computer Society Press, 1994.

[44] Y-L Wu and Malgorzata Marek-Sadowska. An Efficient Router for 2-D Field Programmable Gate Array. In *Proceedings of International Symposium of European Design and Test Conference, 1994. EDAC*, pages 412–416. IEEE, 1994.

[45] Yu-Liang Wu and Malgorzata Marek-Sadowska. Orthogonal Greedy Coupling-A New Optimization Approach to 2-D FPGA routing. In *Proceedings of 32nd International Conference on Design Automation, 1995. DAC'95.*, pages 568–573. IEEE, 1995.

[46] Michael J Alexander and Gabriel Robins. New Performance-Driven FPGA Routing Algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 15(12):1505–1517, 1996.

[47] Mikael Palczewski. Plane Parallel A Maze Router and its Application to FPGAs. In *Proceedings of 29th International ACM/IEEE Design Automation Conference (DAC)*, pages 691–697. IEEE Computer Society Press, 1992.

[48] Stephen Brown, Jonathan Rose, and Zvonko G Vranesic. A Detailed Router for Field-Programmable Gate Arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 11(5):620–628, 1992.

[49] Guy G Lemieux and Stephen D Brown. A Detailed Routing Algorithm for Allocating Wire Segments in Field-Programmable Gate Arrays. In *ACM-SIGDA Physical Design Workshop*, 1993.

[50] Guy GF Lemieux, Stephen D Brown, and Daniel Vranesic. On Two-Step Routing for FPGAs. In *Proceedings of International Symposium on Physical design, 1997*, pages 60–66. ACM, 1997.

[51] Jonathan Greene, Vwani Roychowdhury, Sinan Kaptanoglu, and Abbas El Gamal. Segmented Channel Routing. In *Proceedings of 27th international ACM/IEEE Design Automation Conference (DAC)*, pages 567–572. ACM, 1991.

[52] [Online] FPGA Developers. `http://www.fpgadeveloper.com/2011/07/list-and-comparison-of-fpga-companies.html`.

[53] [Online] sourcetech411. `http://sourcetech411.com/2013/04/top-fpga-companies-for-2013/`.

[54] [Online] Atmel SPLD/CPLD. `http://http://www.atmel.com/products/programmable-logic/spld-cpld/`.

[55] [Online] Microsemi FPGA Families. `http://http://http://www.microsemi.com/products/fpga-soc/fpgas`.

[56] [Online] Abax Product Family Overview. `http://caxapa.ru/thumbs/258641/Abax_ProductBrochure.pdf`.

[57] Ray Bittner, Peter M Athanas, and Mark D Musgrove. Colt: An Experiment in Wormhole Run-Time Reconfiguration. *Journal of SPIE Photonics East96, Boston, MA, USA*, 1996.

[58] Daniel D Gajski, David A. Padua, David J. Kuck, and Robert H. Kuhn. Second Opinion on Data Flow Machines and Languages. *Journal of Computer;(United States)*, 2, 1982.

[59] John L Hennessy and David A Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier, 2012.

[60] John R Hauser and John Wawrzynek. Garp: A MIPS Processor with a Reconfigurable Coprocessor. In *Proceedings of 5th IEEE Symposium on FPGAs for Custom Computing Machines, 1997.*, pages 12–21. IEEE, 1997.

[61] Hartej Singh, Ming-Hau Lee, Guangming Lu, Fadi J Kurdahi, Nader Bagherzadeh, and Eliseu M Chaves Filho. MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications. *IEEE Transactions on Computers*, 49(5):465–481, 2000.

[62] Alan Marshall, Tony Stansfield, Igor Kostarnov, Jean Vuillemin, and Brad Hutchings. A Reconfigurable Arithmetic Array for Multimedia Applications. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays (FPGA)*, pages 135–143. ACM, 1999.

[63] Carl Ebeling, Darren C Cronquist, and Paul Franklin. RaPiDReconfigurable Pipelined Datapath. In *Proceedings of International Symposium on Field-Programmable Logic Smart Applications, New Paradigms and Compilers*, pages 126–135. Springer, 1996.

[64] Don Cherepacha and David Lewis. A Datapath Oriented Architecture for FPGAs. *Proc. FPGA 94, Monterey, CA, USA*, 1994.

[65] Takashi Miyamori and Kunle Olukotun. REMARC: Reconfigurable Multimedia Array Coprocessor. *IEICE Transactions on information and systems*, 82(2):389–397, 1999.

[66] Dev C Chen and Jan M Rabaey. A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic-Specific High-Speed DSP Data Paths. *IEEE Journal of Solid-State Circuits*, 27(12):1895–1904, 1992.

[67] Alfred KW Yeung and Jan M Rabaey. A Reconfigurable Data-driven Multiprocessor Architecture for Rapid Prototyping of High Throughput DSP Algorithms. In *Proceedings of 26th Hawaii International Conference on System Sciences, 1993*, volume 1, pages 169–178. IEEE, 1993.

[68] Jan M Rabaey. Reconfigurable Processing: The Solution to Low-Power Programmable DSP. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP-97.*, volume 1, pages 275–278. IEEE, 1997.

[69] P hw Leong, W Luk, SJE Wilton, S Lopez-Buedo, et al. Virtual Embedded Blocks: A Methodology for Evaluating Embedded Elements in FPGAs. In *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 35–44. IEEE, 2006.

[70] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical Clustering Using Dynamic Modeling. *Computer*, 32(8):68–75, 1999.

[71] Christian Plessl and Marco Platzner. Zippy-A Coarse-Grained Reconfigurable Array With Support for Hardware Virtualization. In *ASAP*, volume 5, pages 213–218, 2005.

[72] M Ander, P Beltrao, B Di Ventura, J Ferkinghoff-Borg, M Foglierini, A Kaplan, C Lemerle, I Tomas-Oliveira, L Serrano, et al. SmartCell, A Framework to Simulate Cellular Processes that Combines Stochastic Approximation with Diffusion and Localisation: Analysis of Simple Networks. *Syst. Biol*, 1(1):129–138, 2004.

[73] Randy T Ong. Programmable Logic Device Which Stores More Than One Configuration and Means for Switching Configurations, June 20 1995. US Patent 5,426,378.

[74] Steven Trimberger, Dean Carberry, Anders Johnson, and Jennifer Wong. A Time-Multiplexed FPGA. In *Proceedings of 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 1997.*, pages 22–28, 1997.

[75] Andre DeHon. Dynamically Programmable Gate Arrays: A Step Toward Increased Computational Density. In *Proceedings of Fourth Canadian Workshop on Field Programmable Devices*, volume 47, 1996.

[76] Weisheng Chong, Sho Ogata, Masanori Hariyama, and Michitaka Kameyama. Architecture of A Multi-Context FPGA Using Reconfigurable Context Memory. In *Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, 2005.*, pages 144a–144a. IEEE, 2005.

[77] Tabula, Technical Support. *Spacetime architecture*, 2010.

[78] Michael Hübner, Diana Göhringer, Juanjo Noguera, and Jürgen Becker. Fast Dynamic and Partial Reconfiguration Data Path With Low Hardware Overhead on Xilinx FPGAs. In *IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010*, pages 1–8. IEEE, 2010.

[79] Kizheppatt Vipin and Suhaib A Fahmy. Automated Partitioning for Partial Reconfiguration Design of Adaptive Systems. In *Proceedings of IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, pages 172–181. IEEE, 2013.

[80] Altera Inc. *SV–5V1: Stratix V Device Handbook*, 2015.

[81] Taro Fujii, K-i Furuta, Masato Motomura, Masahiro Nomura, Masayuki Mizuno, K-i Anjo, Kazutoshi Wakabayashi, Yoshinori Hirota, Y-e Nakazawa, Hiroshi Ito, et al. A Dynamically Reconfigurable Logic Engine With A Multi-Context/Multi-Mode Unified-Cell Architecture. In *Proceedings of IEEE International Conference on Solid-State Circuits, 1999. Digest of Technical Papers. ISSCC.*, pages 364–365, 1999.

[82] [Online] Partial reconfiguration in the ISE design suite. `http://www.xilinx.com/tools/partial-reconfiguration.htm`.

[83] Md Ashfaquzzaman Khan, Naoto Miyamoto, Roel Pantonial, Koji Kotani, Shigetoshi Sugawa, and Tadahiro Ohmi. Improving Multi-Context Execution

Speed on DRFPGAs. In *Solid-State Circuits Conference, 2006. ASSCC 2006. IEEE Asian*, pages 275–278. IEEE, 2006.

[84] Herman Schmit, Michael Butts, Brad L Hutchings, and Steven Teig. Configurable Circuits, IC's, and Systems, September 19 2006. US Patent 7,109,752.

[85] [Online] ABAX2 P-series: Designed for high-performance packet processing. `http://www.tabula.com/products/abax22nm.php`.

[86] Tom R Halfhill. Tabulas time machine. *Microprocessor report*, 131, 2010.

[87] Naoto Miyamoto and Tadahiro Ohmi. A 1.6 mm 2 4,096 Logic Elements Multi-Context FPGA Core in 90nm CMOS. In *Proceedings of IEEE Asian Solid-State Circuits Conference, A-SSCC'08.*, pages 89–92. IEEE, 2008.

[88] Benjamin Gojman, Raphael Rubin, Concetta Pilotto, André DeHon, and Tetsufumi Tanamoto. 3D Nanowire-based Programmable Logic. In *Proceedings of 1st International Conference on Nano-Networks and Workshops, 2006. NanoNet'06.*, pages 1–5. IEEE, 2006.

[89] Dmitri B Strukov and Konstantin K Likharev. CMOL FPGA: A Reconfigurable Architecture for Hybrid Digital Circuits With Two-Terminal Nanodevices. *Journal of Nanotechnology*, 16(6):888, 2005.

[90] Reza MP Rad and Mohammad Tehranipoor. A New Hybrid FPGA with Nanoscale Clusters and CMOS Routing. In *Proceedings of the 43rd Annual Design Automation Conference*, pages 727–730. ACM, 2006.

[91] Edward J Nowak, Ingo Aller, Thomas Ludwig, Keunwoo Kim, Rajiv V Joshi, Ching-Te Chuang, Kerry Bernstein, and Ruchir Puri. Turning Silicon on Its Edge [Double Gate CMOS/FinFET Technology]. *IEEE Circuits and Devices Magazine*, 20(1):20–31, 2004.

[92] Anish Muttreja, Niket Agarwal, and Niraj K Jha. CMOS Logic Design with Independent-Gate FinFETs. In *25th International Conference on Computer Design, 2007. ICCD 2007.*, pages 560–567. IEEE, 2007.

[93] Uygar Avci, Arvind Kumar, Haitao Liu, and Sandip Tiwari. Back-Gated SOI Technology: Power-Adaptive Logic and Non-Volatile Memory Using Identical Processing. In *Proceeding of the 34th European Solid-State Device Research conference, 2004. ESSDERC 2004.*, pages 285–288. IEEE, 2004.

[94] Kaushik Roy, Hamid Mahmoodi, Saibal Mukhopadhyay, Hari Ananthan, Aditya Bansal, and Tamer Cakici. Double-gate SOI Devices for Low-Power and High-Performance Applications. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 217–224. IEEE Computer Society, 2005.

[95] Paul Beckett. A Fine-Grained Reconfigurable Logic Array Based on Double Gate Transistors. In *IEEE International Conference on Field-Programmable Technology, 2002.(FPT).*, pages 260–267. IEEE, 2002.

[96] Paul Beckett. A Low-Power Reconfigurable Logic Array Based on Double-Gate Transistors. *IEEE transactions on very large scale integration (VLSI) systems*, 16(2):115–123, 2008.

[97] Wei Zhang, Niraj K Jha, and Li Shang. Low-Power 3D Nano/CMOS Hybrid Dynamically Deconfigurable Architecture. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 6(3):10, 2010.

[98] [Online] Comparison of the DSP Blocks in Stratix Series FPGAs. `https://www.altera.com/products/fpga/features/stx-dsp-block.html`.

[99] [Online] Stratix V DSP Block. `https://www.altera.com/products/fpga/features/dsp/stratix-v-dsp-block.html`.

[100] [Online] Stratix 10: The Most Powerful, Most Efficient FPGA for Signal Processing. `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/backgrounder/stratix10-floating-point-backgrounder.pdf`.

[101] [Online] Comparison of the DSP Blocks in Stratix Series FPGAs. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/sg/product-catalog.pdf.

[102] [Online] Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet. http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf.

[103] [Online] Ultra Scale Architecture DSP Slice. http://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf.

[104] [Online] QUIP. http://www.xilinx.com/technology/dsp/xtremedsp.htm.

[105] Ravi K Kolagotla, Jose Fridman, Bradley C Aldrich, Marc M Hoffman, William C Anderson, Michael S Allen, David B Witt, Randy R Dunton, and Lawrence A Booth Jr. High Performance Dual-MAC DSP Architecture. *IEEE Transactions of Signal Processing Magazine*, 19(4):42–53, 2002.

[106] Matthias H Weiss, Frank Engel, and Gerhard P Fettweis. A New Scalable DSP Architecture for System on Chip (SoC) Domains. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, 1999.*, volume 4, pages 1945–1948. IEEE, 1999.

[107] Hadi Parandeh-Afshar and Paolo Ienne. Highly Versatile DSP Blocks for Improved FPGA Arithmetic Performance. In *Proceedings of 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2010.*, pages 229–236. IEEE, 2010.

[108] Mentor Graphics. Modelsim simulator, 2015.

[109] Robert Francis, Jonathan Rose, and Zvonko Vranesic. Chortle-crf: Fast Technology Mapping For LookUp Table-based FPGAs. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pages 227–233. ACM, 1991.

[110] Robert Francis, Jonathan Rose, and Zvonko Vranesic. Chortle-crf: Fast Technology Mapping for LookUp Table-based FPGAs. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pages 227–233. ACM, 1991.

[111] Robert J Francis, Jonathan Rose, and Zvonko Vranesic. Technology Mapping of Lookup Table-based FPGAs for Performance. In *Proceedings of IEEE International Conference on Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991*, pages 568–571. IEEE, 1991.

[112] Jason Cong and Yuzheng Ding. FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in LookUp-Table Based FPGA Designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(1):1–12, 1994.

[113] Jason Cong, Hui Huang, and Xin Yuan. Technology Mapping and Architecture Evalution for k/m-macrocell-based FPGAs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 10(1):3–23, 2005.

[114] Amir H Farrahi and Majid Sarrafzadeh. FPGA Technology Mapping for Power Minimization. In *Proceedings of International Conference on Field-Programmable Logic Architectures, Synthesis and Applications*, pages 66–77. Springer, 1994.

[115] Martine Schlag, Jackson Kong, and Pak K Chan. Routability-Driven Technology Mapping for LookIp Table-based FPGA's. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(1):13–26, 1994.

[116] Narasimha B Bhat and Dwight D Hill. Routable Technology Mapping for LUT FPGAs. In *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1992. ICCD'92.*, pages 95–98. IEEE, 1992.

[117] Jason Cong and Yuzheng Ding. Combinational Logic Synthesis for LUT based Field Programmable Gate Arrays. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 1(2):145–204, 1996.

[118] Deming Chen, Jason Cong, and Peichen Pan. FPGA Design Automation: A survey. *Foundations and Trends in Electronic Design Automation*, 1(3):139–169, 2006.

[119] Vaughn Betz and Jonathan Rose. VPR: A New Packing, Placement and Routing Tool for FPGA Research. In *Proceedings of the International Conference on Field-Programmable Logic and Applications (FPL)*, pages 213–222, 1997.

[120] Pongstorn Maidee, Cristinel Ababei, and Kia Bazargan. Timing-driven Partitioning-based Placement for Island Style FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*, 24(3):395–406, 2005.

[121] Larry McMurchie and Carl Ebeling. PathFinder: A Negotiation-based Performance-driven Router for FPGAs. In *Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays*, pages 111–117. ACM, 1995.

[122] Stephen Brown, Jonathan Rose, and Zvonko G Vranesic. A Detailed Router for Field-Programmable Gate Arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(5):620–628, 1992.

[123] Guy G Lemieux and Stephen D Brown. A Detailed Routing Algorithm for Allocating Wire Segments in Field-programmable Gate Arrays. In *ACM-SIGDA Physical Design Workshop*, 1993.

[124] Vaughn Betz and Jonathan Rose. VPR: A New Packing, Placement and Routing tool for FPGA Research. In *Proceedings of International Symposium on Field-Programmable Logic and Applications (FPL)*, pages 213–222. Springer Berlin Heidelberg, 1997.

[125] Ian Kuon and Jonathan Rose. Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):203–215, 2007.

[126] Steven JE Wilton, Su-Shin Ang, and Wayne Luk. The Impact of Pipelining on Energy Per Operation in Field-Programmable Gate Arrays. In *Proceedings of International Conference on Field Programmable Logic and Application*, pages 719–728. Springer, 2004.

[127] William G Osborne, Wayne Luk, José Gabriel F Coutinho, and Oskar Mencer. Reconfigurable Design With Clock Gating. In *ICSAMOS*, pages 187–194. Citeseer, 2008.

[128] William G Osborne, Jose Coutinho, Wayne Luk, and Oskar Mencer. Power-Aware and Branch-Aware Word-Length Optimization. In *Proceedings of 16th International Symposium on Field-Programmable Custom Computing Machines, FCCM'08*, pages 129–138. IEEE, 2008.

[129] Partha Biswas, Sudarshan Banerjee, Nikil Dutt, Paolo Ienne, and Laura Pozzi. Performance And Energy Benefits of Instruction Set Extensions in an FPGA Soft Core. In *Proceedings of 19th International Conference on VLSI Design, held jointly with 5th International Conference on Embedded Systems and Design, 2006.*, pages 6–pp. IEEE, 2006.

[130] Robert G Dimond, Oskar Mencer, and Wayne Luk. Combining Instruction Coding and Scheduling to Optimize Energy in System-on-FPGA. In *Proceedings of 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2006. FCCM'06.*, pages 175–184. IEEE, 2006.

[131] Maurice Meijer, Rohini Krishnan, and Martijn Bennebroek. Energy-Efficient FPGA Interconnect Design. In *Proceedings of the conference on Design, automation and test in Europe (DATE): Designers' forum*, pages 42–47. European Design and Automation Association, 2006.

[132] Satish Sivaswamy, Gang Wang, Cristinel Ababei, Kia Bazargan, Ryan Kastner, and Eli Bozorgzadeh. HARP: Hard-Wired Routing Pattern FPGAs.

In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-Programmable Gate Arrays (FPGA)*, pages 21–29. ACM, 2005.

[133] Jason H Anderson and Farid N Najm. A novel low-power FPGA routing switch. In *Proceedings of IEEE International Conference on Custom Integrated Circuits 2004.*, pages 719–722. IEEE, 2004.

[134] Yan Lin, Fei Li, and Lei He. Routing Track Duplication With Fine-Grained Power-Gating for FPGA Interconnect Power Reduction. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pages 645–650. ACM, 2005.

[135] Eric Kusse and Jan Rabaey. Low-Energy Embedded FPGA Structures. In *Proceedings of International Symposium on Low Power Electronics and Design, 1998.*, pages 155–160. IEEE, 1998.

[136] Sami Khawam, Ioannis Nousias, Mark Milward, Ying Yi, Mark Muir, and Tughrul Arslan. The Reconfigurable Instruction Cell Array. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*, 16(1):75–85, 2008.

[137] Warren Miller and Kirk Owyang. Designing a High Performance FPGA-Using the PREP Benchmarks. In *Proceedings of International Conference Record WESCON/'93.*, pages 234–239. IEEE, 1993.

[138] Min Xu and Fadi Kurdahi. Area and Timing Estimation for LookUp Table Based FPGAs. In *Proceedings of the 1996 European conference on Design and Test*, page 151. IEEE Computer Society, 1996.

[139] Min Xu and Fadi J Kurdahi. Layout-driven RTL Binding Techniques for High-Level Synthesis Using Accurate Estimators. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2(4):312–343, 1997.

[140] Rolf Enzler, Tobias Jeger, Didier Cottet, and Gerhard Tröster. High-level Area and Performance Estimation of Hardware Building Blocks on FPGAs. In *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, pages 525–534. Springer, 2000.

[141] Anshuman Nayak, Malay Haldar, Alok Choudhary, and Prithviraj Banerjee. Accurate Area and Delay Estimators for FPGAs. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, page 862. IEEE Computer Society, 2002.

[142] Dhananjay Kulkarni, Walid A Najjar, Robert Rinker, and Fadi J Kurdahi. Fast area estimation to support compiler optimizations in FPGA-based reconfigurable systems. In *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on*, pages 239–247. IEEE, 2002.

[143] Altera. *APEX 20K Programmable Logic Device Family*, 1990.

[144] Byoungro So, Pedro C Diniz, and Mary W Hall. Using Estimates from Behavioral Synthesis Tools in Compiler-Directed Design Space Exploration. In *Proceedings of the 40th annual Design Automation Conference*, pages 514–519. ACM, 2003.

[145] KR Shesha Shayee, Joonseok Park, and Pedro C Diniz. Performance and Area Modeling of Complete FPGA Designs in the Presence of Loop Transformations. In *Field Programmable Logic and Application*, pages 313–323. Springer, 2003.

[146] Sumit Mohanty, Seonil Choi, Ju-wook Jang, and Viktor K Prasanna. A Model-based Methodology for Application Specific Energy Efficient Data Path Design Using FPGAs. In *Proceedings of IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2002.*, pages 76–87. IEEE, 2002.

[147] Deming Chen, Jason Cong, and Yiping Fan. Low-Power High-Level Synthesis for FPGA Architectures. In *Proceedings of International symposium on Low Power Electronics and Design*, pages 134–139. ACM, 2003.

[148] Francis G Wolff, Michael J Knieser, Dan J Weyer, and Chris A Papachristou. High-Level Low Power FPGA Design Methodology. In *Proceedings of*

the IEEE National Aerospace and Electronics Conference, NAECON 2000., pages 554–559. IEEE, 2000.

[149] Deming Chen, Jason Cong, and Junjuan Xu. Optimal Module and Voltage Assignment for Low-Power. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 850–855. ACM, 2005.

[150] Michael J Alexander. Power Optimization for FPGA Look-Up Tables. In *Proceedings of International symposium on Physical design*, pages 156–162. ACM, 1997.

[151] Jason H Anderson and Farid N Najm. Active Leakage Power Optimization for FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*, 25(3):423–437, 2006.

[152] Deming Chen, Jason Cong, Fei Li, and Lei He. Low-Power Technology Mapping for FPGA Architectures With Dual Supply Voltages. In *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 109–117. ACM, 2004.

[153] Julien Lamoureux and Steven JE Wilton. On the Interaction Between Power-Aware FPGA CAD Algorithms. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 701. IEEE Computer Society, 2003.

[154] Chua-Chin Wang and Cheng-Pin Kwan. Low power Technology Mapping by Hiding High-Transition Paths in Invisible Edges for LUT-Based FPGAs. In *Proceedings of IEEE International Symposium onCircuits and Systems, 1997. ISCAS'97.*, volume 3, pages 1536–1539. IEEE, 1997.

[155] Zhi-Hong Wang, En-Cheng Liu, Jianbang Lai, and Ting-Chi Wang. Power Minization in LUT-based FPGA Technology Mapping. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, pages 635–640. ACM, 2001.

[156] Deming Chen and Jason Cong. Delay Optimal Low-Power Circuit Clustering for FPGAs With Dual Supply Voltages. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design, 2004. ISLPED'04.*, pages 70–73. IEEE, 2004.

[157] Hassan Hassan, Mohab Anis, Antoine El Daher, and Mohamed Elmasry. Activity packing in FPGAs for leakage power reduction. In *Proceedings of International Symposium on Design, Automation and Test in Europe, (DATE) 2005.*, pages 212–217. IEEE, 2005.

[158] Amit Singh, Ganapathy Parthasarathy, and Malgorzata Marek-Sadowska. Efficient Circuit Clustering for Area and Power Reduction in FPGAs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 7(4):643–663, 2002.

[159] Balakrishna Kumthekar and Fabio Somenzi. Power and Delay Reduction via Simultaneous Logic and Placement Optimization in FPGAs. In *Proceedings of the conference on Design, automation and test in Europe*, pages 202–207. ACM, 2000.

[160] Kaushik Roy. Power-dissipation driven FPGA place and route under timing constraints. *IEEE Transactions on Circuits and Systems I (TCAS-I): Fundamental Theory and Applications*, 46(5):634–637, 1999.

[161] Nozomu Togawa, Kaoru Ukai, Masao Yanagisawa, and Tatsuo Ohtsuki. A Simultaneous Placement and Global Routing Algorithm for FPGAs With Power Optimization. In *Proceedings of IEEE Asia-Pacific Conference on Circuits and Systems, 1998. (APCCAS)*, pages 125–128. IEEE, 1998.

[162] Russell Tessier, Vaughn Betz, David Neto, Aaron Egier, and Thiagaraja Gopalsamy. Power-Efficient RAM Mapping Algorithms for FPGA Embedded Memory Blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*, 26(2):278–290, 2007.

[163] Wei Zhang, Niraj K Jha, and Li Shang. NATURE: A Hybrid Nanotube/CMOS Dynamically Reconfigurable Architecture. In *Proceedings of the 43rd*

International Annual Design Automation Conference (DAC), pages 711–716. ACM, 2006.

[164] Scott Hauck, Thomas W Fry, Matthew M Hosler, and Jeffrey P Kao. The Chimaera Reconfigurable Functional Unit. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(2):206–217, 2004.

[165] Pierre G Paulin and John P Knight. Force-directed Scheduling for the Behavioral Synthesis of ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(6):661–679, 1989.

[166] [Online] Scheduling. `http://en.wikipedia.org/wiki/Scheduling/`.

[167] [Online] VPR and T-VPack: Versatile Packing,Placement and Routing for FPGAs. `http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html`.

[168] Ian Kuon, Russell Tessier, and Jonathan Rose. FPGA Architecture: Survey and Challenges. *Foundations and Trends® in Electronic Design Automation*, 2(2):135–253, 2008.

[169] Reiner Hartenstein. A Decade of Reconfigurable Computing: A Visionary Retrospective. In *Proceedings of International Conference on Design, automation and test in Europe (DATE)*, pages 642–649. IEEE Press, 2001.

[170] [Online] Xilinx. `http://www.xilinx.com/`.

[171] [Online] Altera. `http://www.altera.com/`.

[172] Tung Thanh Hoang, M Sjalander, and Per Larsson-Edefors. High-speed, Energy-efficient 2-Cycle Multiply-Accumulate Architecture. In *Proceedings of IEEE International SOC Conference (SOCC) 2009.*, pages 119–122, 2009.

[173] Magnus Själander and Per Larsson-Edefors. High-speed and Low-power Multipliers Using the Baugh-Wooley Algorithm and HPM Reduction Tree. In *Proceedings of the International Conference on Electronics, Circuits and Systems (ICECS)*, pages 33–36, 2008.

[174] Christopher S Wallace. A Suggestion for a Fast Multiplier. *IEEE Transactions on Electronic Computers*, (1):14–17, 1964.

[175] Henrik Eriksson, Per Larsson-Edefors, Mary Sheeran, Magnus Själander, Daniel Johansson, and M Scholin. Multiplier Reduction Tree with Logarithmic Logic Depth and Regular Connectivity. In *Proceedings of IEEE International Conference on Circuits and Systems (ISCAS)*, volume 6, 2006.

[176] Tung Thanh Hoang, Magnus Sjalander, and Per Larsson-Edefors. A High-Speed, Energy-Efficient 2C MAC architecture and Its application to a double-throughput MAC unit. *IEEE Transaction on Circuits and Systems-I TCAS I*, 57(12):3073–3081, 2010.

[177] Xilinx Inc. *XAPP467: Using Embedded Multipliers in Spartan-3 FPGAs*, 2003.

[178] Altera Inc. *AN306: Implementing Multipliers in FPGA Devices*, 2004.

[179] Vaughn Betz and Jonathan Rose. FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density. In *Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA)*, pages 59–68, 1999.

[180] André DeHon and Raphael Rubin. Design of fpga interconnect for multilevel metallization. *IEEE transactions on very large scale integration (VLSI) systems*, 12(10):1038–1050, 2004.

[181] Elaheh Bozorgzadeh, Seda Ogrenci-Memik, and Majid Sarrafzadeh. RPack: Routability-Driven Packing for Cluster-Based FPGAs. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, pages 629–634. ACM, 2001.

[182] Xilinx Inc. *UG431: XtremeDSP DSP48A for Spartan-3A DSP FPGAs, User Guide*, 2011.

[183] [Online] Finite Impulse Response. `http://en.wikipedia.org/wiki/Finite_impulse_response/`.

[184] John G Proakis. *Digital Signal Processing: Principles, Algorithms, and Applications, 4/e*. Pearson Education India, 2007.

[185] Chao Cheng and Keshab K Parhi. Hardware Efficient Fast Parallel FIR Filter Structures based on Iterated Short Convolution. In *Proceedings of International Symposium on Circuits and Systems, 2004. ISCAS'04.*, volume 3, pages III–361. IEEE, 2004.

[186] [Online] Finite Impulse Response. `http://en.wikipedia.org/wiki/Linear_phase/`.

[187] Fons Bruekers. Symmetry and Efficiency in Complex FIR Filters. *Philips Research Laboratories, Eindhoven*, 2009.

[188] LIM YONG-CHING and AG Constantinides. Linear Phase FIR Digital Filter Without Multipliers'. In *Proceedings of International Symposium of Circuits and Systems (ISCAS)*, 1979.

[189] Miroslav Vlcek, Pavel Zahradnik, and Rolf Unbehauen. Analytical Design of FIR Filters. *IEEE Transactions on Signal Processing*, 48(9):2705–2709, 2000.

[190] Kenneth Steiglitz, Thomas W Parks, and James F Kaiser. METEOR: A Constraint-based FIR Filter Design Program. *IEEE Transactions on Signal Processing*, 40(8):1901–1909, 1992.

[191] Loganathan Lingappan, Srivaths Ravi, and Niraj K Jha. Satisfiability-Based Test Generation for Nonseparable RTL Controller-Datapath Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(3):544–557, 2006.

[192] Indradeep Ghosh, Anand Raghunathan, and Niraj K Jha. Hierarchical Test Generation and Design for Testability Methods for ASPPs and ASIPs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(3):357–370, 1999.

[193] Chunho Lee, Miodrag Potkonjak, and William H Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *Proceedings of 30th annual ACM/IEEE International Symposium on Microarchitecture*, pages 330–335, 1997.

[194] Seung Eun Lee and Nader Bagherzadeh. A Variable Frequency Link for a Power-Aware Network-on-Chip (NoC). *INTEGRATION, the VLSI journal*, 42(4):479–485, 2009.

[195] [Online] Power Analysis Estimation of a Digital Design: A Quick Tutorial:. http://http://www.vlsiip.com/power/, 2011.

[196] Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matthew Moe, and R Reed Taylor. PipeRench: A reconfigurable Architecture and Compiler. *Computer*, 33(4):70–77, 2000.

[197] Altera Inc. *Variable Precision DSP Blocks in Stratix V Devices*, 2013.

[198] Masayuki Ito, David Chinnery, and Kurt Keutzer. Low Power Multiplication Algorithm for Switching Activity Reduction Through Operand Decomposition. In *Proceedings of Internat Conference on Computer Design, 2003.*, pages 21–26, 2003.

[199] R Sakthivel, K Sravanthi, and Harish M Kittur. Low Power Energy Efficient Pipelined Multiply-Accumulate Architecture. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, pages 226–231, 2012.

[200] Vasily G Moshnyaga. Reducing Switching Activity of Subtraction via Variable Truncation of the Most-Significant Bits. *Journal of VLSI signal processing systems for Signal, Image and Video Technology*, 33(1-2):75–82, 2003.

[201] H S Krishnaprasad Puttam, P Sivadurga Rao, and NVG Prasad. Implementation of Low Power and High Speed Multiplier-Accumulator Using SPST Adder and Verilog. *International Journal of Modern Engineering Research (IJMER)*, 2(5), 2012.

[202] Tung Thanh Hoang, M Sjalander, and Per Larsson-Edefors. High-speed, Energy-Efficient 2-Cycle Multiply-Accumulate Architecture. In *Proceedings of International Conference on System on Chip SOC*. IEEE, 2009.

[203] B. Ronak and S.A. Fahmy. Evaluating the Efficiency of DSP Block Synthesis Inference from Flow Graphs. In *Proceedings of Field Programmable Logic and Applications (FPL)*, pages 727 –730, Aug 2012.

[204] Thomas K Callaway and Earl E Swartzlander Jr. Power-Delay Characteristics of CMOS Multipliers. In *Proceedings of International Symposium on Computer Arithmetic*, pages 26–32, 1997.

[205] Srihari Cadambi, Jeffrey Weener, Seth Copen Goldstein, Herman Schmit, and Donald E Thomas. Managing Pipeline-Reconfigurable FPGAs. In *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays (FPGA)*, pages 55–64. ACM, 1998.

[206] Deepali Deshpande, Arun K Somani, and Akhilish Tyagi. Configuration Caching vs Data Caching for Striped FPGAs. In *Proceedings of the 1999 ACM/SIGDA seventh International Symposium on Field programmable Gate Arrays (FPGA)*, pages 206–214, 1999.

[207] Rakesh Warrier, CH Vun, and Wei Zhang. A Low-Power Pipelined MAC Architecture Using Baugh-Wooley Based Multiplier. In *Proceedings of IEEE 3rd Global Conference on Consumer Electronics (GCCE), 2014.*, pages 505–506. IEEE, 2014.

[208] R Mahesh and A Prasad Vinod. New Reconfigurable Architectures for Implementing FIR Filters With Low Complexity. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*, 29(2):275–288, 2010.

[209] Abhishek Ambede, KG Smitha, and A Prasad Vinod. Flexible Low Complexity Uniform and Nonuniform Digital Filter Banks With High Frequency Resolution for Multistandard Radios. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*, 23(4):631–641, 2015.

[210] Sumedh Dhabu, Kavallur Gopi Smitha, and A Prasad Vinod. Design of Reconfigurable Filter Bank Architecture Using Improved Coefficient Decimation-Interpolation-Masking Technique for Multi-Standard Wireless Communication Receivers. *Journal of Low Power Electronics*, 10(3):417–428, 2014.

[211] Ting-Jung Lin, Wei Zhang, and Niraj K Jha. FDR: A Fine-Grain Dynamically Reconfigurable Architecture Aimed at Reducing the FPGA-ASIC Gaps. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(12):2607–2620, 2014.

[212] B. Ronak and S. A. Fahmy. Evaluating the Efficiency of DSP Block Synthesis Inference From Flow Graphs. In *Proceedings of International Conference on Field Programmable Logic and Applications (FPL)*, pages 727–730, 2012.

[213] M. Hatamian. A 70-MHz 8-bit x 8-bit Parallel Pipelined Multiplier in 2.5-µm CMOS. *IEEE Journal on Solid-State Circuits*, 21(4):505–513, 1986.

[214] M Machhout, M Zeghid, W El Hadj Youssef, B Bouallegue, A Baganne, and R Tourki. Efficient large Numbers Karatsuba-Ofman Multiplier Designs for Embedded Systems. *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering*, 3(4), 2009.

[215] Xu, Simin and Fahmy, Suhaib A. and McLoughlin, Ian V. Square-Rich Fixed Point Polynomial Evaluation on FPGAs. In *Proceedings of International Symposium on Field-programmable Gate Arrays (FPGA)*, pages 99–108. ACM, 2014.

[216] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.

[217] Daniel D Gajski, Nikil Dutt, Allen Wu, and Steve Lin. High Level Synthesis, Introduction to Chip and System Design, chapter 1, 1992.

[218] George A Constantinides. Word-Length Optimization for Differentiable Nonlinear Systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 11(1):26–43, 2006.

[219] Chun Tak Chow, Lai Suen Mandy Tsui, Philip Heng Wai Leong, Wayne Luk, and Steven JE Wilton. Dynamic Voltage Scaling for Commercial FPGAs. In *Proceedings of IEEE International Conference on Field-Programmable Technology, 2005.*, pages 173–180. IEEE, 2005.

[220] Fei Li, Yan Lin, Lei He, Deming Chen, and Jason Cong. Power Modeling and Characteristics of Field Programmable Gate Arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(11):1712–1724, 2005.

[221] Norman Richard Draper, Harry Smith, and Elizabeth Pownell. *Applied Regression Analysis*, volume 3. Wiley New York, 1966.

[222] Thomas H Cormen. *Introduction to Algorithms*. MIT press, 2009.