Efficient Architectures and Power Modelling of Multiresolution Analysis Algorithms on FPGA

A thesis submitted for the degree of
Doctor of Philosophy

by

Abdul Naser Sazish
B.E.

Brunel
UNIVERSITY
WEST LONDON

Electronic and Computer Engineering
School of Engineering and Design
Brunel University, West London

February 2011

Abstract

In the past two decades, there has been huge amount of interest in Multiresolution Analysis Algorithms (MAAs) and their applications. Processing some of their applications such as medical imaging are computationally intensive, power hungry and requires large amount of memory which cause a high demand for efficient algorithm implementation, low power architecture and acceleration. Recently, some MAAs such as Finite Ridgelet Transform (FRIT) Haar Wavelet Transform (HWT) are became very popular and they are suitable for a number of image processing applications such as detection of line singularities and contiguous edges, edge detection (useful for compression and feature detection), medical image denoising and segmentation. Efficient hardware implementation and acceleration of these algorithms particularly when addressing large problems are becoming very challenging and consume lot of power which leads to a number of issues including mobility, reliability concerns. To overcome the computation problems, Field Programmable Gate Arrays (FPGAs) are the technology of choice for accelerating computationally intensive applications due to their high performance. Addressing the power issue requires optimisation and awareness at all level of abstractions in the design flow.

The most important achievements of the work presented in this thesis are summarised here.

Two factorisation methodologies for HWT which are called HWT Factorisation Method1 and (HWTFM1) and HWT Factorasation Method2 (HWTFM2) have been explored to increase number of zeros and reduce hardware resources. In addition, two novel efficient and optimised architectures for proposed methodologies based on Distributed Arithmetic (DA) principles have been proposed. The evaluation of the architectural results have shown that the proposed architectures results have reduced the arithmetics calculation (additions/subtractions) by 33% and 25% respectively compared to direct implementation of HWT and outperformed existing results in place. The proposed HWTFM2 is implemented on advanced and low power FPGA devices using Handel-C language. The FPGAs implementation results have outperformed other existing results in terms of area and maximum frequency. In addition, a novel efficient architecture for Finite Radon Transform (FRAT) has also been proposed. The proposed architecture is integrated with the developed HWT architecture to build an optimised architecture for FRIT. Strategies such as parallelism and pipelining have been deployed at the architectural level for efficient implementation on different FPGA devices. The proposed FRIT architecture performance

has been evaluated and the results outperformed some other existing architecture in place. Both FRAT and FRIT architectures have been implemented on FPGAs using Handel-C language. The evaluation of both architectures have shown that the obtained results outperformed existing results in place by almost 10% in terms of frequency and area. The proposed architectures are also applied on image data (256×256) and their Peak Signal to Noise Ratio (PSNR) is evaluated for quality purposes.

Two architectures for cyclic convolution based on systolic array using parallelism and pipelining which can be used as the main building block for the proposed FRIT architecture have been proposed. The first proposed architecture is a linear systolic array with pipelining process and the second architecture is a systolic array with parallel process. The second architecture reduces the number of registers by 42% compare to first architecture and both architectures outperformed other existing results in place. The proposed pipelined architecture has been implemented on different FPGA devices with vector size (N) 4,8,16,32 and word-length (W=8). The implementation results have shown a significant improvement and outperformed other existing results in place.

Ultimately, an in-depth evaluation of a high level power macromodelling technique for design space exploration and characterisation of custom IP cores for FPGAs, called functional level power modelling approach have been presented. The mathematical techniques that form the basis of the proposed power modeling has been validated by a range of custom IP cores. The proposed power modelling is scalable, platform independent and compares favorably with existing approaches. A hybrid, top-down design flow paradigm integrating functional level power modelling with commercially available design tools for systematic optimisation of IP cores has also been developed. The in-depth evaluation of this tool enables us to observe the behavior of different custom IP cores in terms of power consumption and accuracy using different design methodologies and arithmetic techniques on virous FPGA platforms. Based on the results achieved, the proposed model accuracy is almost 99% true for all IP core's Dynamic Power (DP) components.

Certificate of Originality

I hereby certify that the work presented in this thesis is my original research and has not been presented for a higher degree at any other university or institute.

Signed:---------- Dated: ----------

(Abdul Naser Sazish)

To my parents

Acknowledgements

First off all I would like to take the opportunity to thank my family for their endless support whenever the need arose. A very special thank goes to my mother and my father for their morale and financial support from miles away throughout the difficult days of my life. I could not ask a better mum and dad than them. This thesis is dedicated to my mother's memory who I have lost during the last stage of my research work and can not be replace with anything. Also I would like thank my brothers (specially Salih Sazish) and sisters for their support and being with me throughout my study. I would like to thank H. Halim for her endless moral support whenever I need it during my research.

The completion of this PhD would not have been possible, without the constant support, encouragement, motivation and most importantly constructive criticism provided by my supervisor, Dr. Abbes Amira. I greatly value the opportunity of working with him. The only gift I could give to him in return is a very special thank. I would like to take the opportunity to thank Dr Maysam Abbod for his positive and constructive feedback throughout my research.

I would like to thank Thomas Gerald Gray Charitable Trust for the financial support over the period of this research and all the support staff in the School of Engineering and Design for their help and assistance.

Finally, I would like to thank Miss J. Markechova for her caring and passionate support throughout my studying. last but not less, I would like to thank all my friends and family, specially my colleagues who I shared office with during this period. It was a great pleasure to work with them.

Author's Publication

Journal Papers (Submitted)

1. A. N. Sazish, M. S. Sharif and A. Amira, "Efficient FPGA Implementation of HWT using Sparse Matrix Factorisation for Medical Imaging" *Elsevier Journal of Digital Signal Processing (DSP)*

2. A. N. Sazish, A. Amira, "Architecture Level Optimization and Efficient FPGA Implementation of Finite Ridgelet Transform" *Special Issue on Embedded System Implementation Using Reconfigurable Hardware, EURASIP Journal on Embedded Systems*

3. A. N. Naser, S. Chandrasekaran, and A. Amira "High level Power Modelling Technique for Custom IP Cores on FPGAs. *IET Computer and Digital Techniques.*

Conference Papers (Accepted)

1. A. N. Sazish and A. Amira, "An efficient architecture for HWT using Sparse Matrix Factorisation and DA Principles," *IEEE Asia Pacific Conference on Circuits and Systems, 2008. APCCAS 2008*, Dec 2008, pp. 1308-1311 Macau.

2. M. S. Sharif, Abdul N. Sazish, and A. Amira "An Efficient Algorithm and Architecture for Medical Image Segmentation and Tumour Detection" *IEEE Biomedical Circuits and Systems Conference (BIOCAS-2008)*, November 20-22, 2008, Baltimore, HD. USA.

3. A. N. Sazish, S. Chandrasekaran, and A. Amira, "Efficient Systolic Architecture and Power Modeling for Finite Ridgelet Transform," *IEEE 12th International Confer-*

ence on Computer Vision Workshops (ICCV Workshops), Oct. 2009, pp. 821-827 Japan.

4. A. N. Sazish, M. Sharif, and A. Amira, "Hardware Implementation and Power Analysis of HWT for Medical Imaging," *16th IEEE International Conference on Electronics, Circuits, and Systems, 2009 ICECS 2009*, Dec. 2009, pp. 775-778, Tunisia.

5. H. Taha, A. N. Sazish, A. Ahmad, M. S. Sharif, and A. Amira "Efficient FPGA Implementation of a Wireless Communication System Using Bluetooth Connectivity", *IEEE International Conference on Circuits and Systems (ISCAS 2010), Proceedings of May 2010* Paris, France.

Contents

List of Figures

List of Tables

List of Abbreviations

AC : Area Complexity

AG : Address Generator

ALI : Address Logic Initialiser

ASIC : Application Specific Integrated Circuit

CAFD : Cycle Accurate Functional Description

CDFG : Control-Data Flow Graph

CDMA : Code Division Multiple Access

CLB : Configurable Logic Block

CMOS : Complementary Metal Oxide Semiconductor

CP : Clock Power

CT: Computed Tomography

CWT : Continues Wavelet Transform

CPLD : Complex Programmable Logic Device

CAD : Computer Aid Design

DA : Distributed Arithmetic

DBWT : Discrete Bi-orthogonal Wavelet Transform

DCT : Discrete Cosine Transform

DICOM: Digital Imaging and Communications in Medicine

DFF : D Flip Flop

DFG : Data Flow Graph

DFT : Discrete Fourier Transform

DHT : Discrete Hartley Transform

DOT : Discrete Orthogonal Transform

DES : Data Encryption standard

DP : Dynamic Power

DSP : Digital Signal Processing

EDIF : Electronic Design Interchange Format

FFT : Fast Fourier Transform

FHT : Fast Hadamard Transform

FMAT : FPGA MATrix Algorithms

FPGAs : Field Programmable Gate Arrays

FRAT : Finite Radon Transform

FRIT : Finite Ridgelet Transform

GMM : Gaussian Mixture Modelling

GPU : Graphic Processing Unit

GPP : General Purpose Processor

HDL : Hardware Definition Language

HLS : High Level Synthesis

HDTV : High Definition TeleVision

HT : Hadamard Transform

HWT : Haar Wavelet Transform

HWTM1 : Haar Wavelet Transform Method 1

HWTM2 : Haar Wavelet Transform Method 2

I/O : Input/Output

IC : Integrated Circuit

IP : Intellectual Property

InP : Input Power

IVSL : Image and Vision Systom Laboratory

JPEG: Joint Photographic Experts Group

LE : Logic Element

LFSR : Linear Feedback Shift Registers

LP : Logic Power

LPW : Linear PieceWise

LSB : Least Significant Bit

LUT : Look Up Table

MB : Memory Block

MAAs : Multiresolution Analysis Algorithms

MPGAs : Mask Programable Gate Arrays

MRI : Magnetic Resonance Imaging

MSB : Most Significant Bit

MIMO : Multiple Input Multiple Output

OP : Output Power

PACE : Pinout Area Constraints Editor

PAL : Programmable Array Logic

PAR : Place and Route

PDA : Personal Digital Assistant

PE : Processing Element

PLD : Programmable Logic Device

PSNR : Peak Signal to Noise Ratio

RAM : Random Access Memory

ROM : Read Only Memory

RT : Radon Transform

RTL : Register Transfer Level

RAG : Reconfigurable Address Generator

SIPOSR : Serial-In Parallel-Out Shift Register

SoC : System on Chip

SoPC : System on a Programmable Chip

SP : Signal Power

SRAM : Static Random Access Memory

TC : Time Complexity

TDM : Time Division Multiplexing

VPR : Versatile Place and Route

W-H : Walsh-Hadamard

WHT : Walsh-Hadamard Transform

XCL : Xilinx Coregen Library

Chapter 1

Introduction

Fast advancement in the world of technology specially in the field of digital image processing brings new challenges everyday and leaves the researchers to search for a better solution. In the past decades digital image and signal processing is one of the fastest growing areas of the electronics industry. These technologies are having an increasing impact in a wide variety of application areas such, wireless communications, telecommunication, biometrics, biomedical imaging, multimedia indexing storage, computer vision and remote sensing.

The characterisation of signal processing has changed in many applications, especially those involving sophisticated algorithms that require real-time processing of high volumes of data. Increasing demands for faster and more sophisticated processing show absolutely no sign of failing for signal processing applications. Therefore, appropriate algorithms and high performance systems are required by the developers for fast computations of these applications.

Advanced image and signal processing techniques in a wide range of disciplines and applications, from computer vision and medical imaging to image and video compression, are replacing previous generations' technology offering enhancements, such as better streaming capability, higher compression for a given quality and lower latency. This can be seen from the considerable amount of literature on the subject, including major international conferences and emerging standards such as JPEG2000 for image compression [36], and Digital Imaging and Communications in Medicine (DICOM) for 3D medical imaging [37]. Researchers are working round the clock to develop efficient algorithms and architectures suitable for these applications. Therefore, application designers face many new and diffi-

cult challenges as they attempt to deploy technology that can execute high-performance computations, manipulate larger and larger data sets and better visualise increasingly complex data. Among these algorithms, multiresolution analysis one of them to be considered carefully due to the huge interest in recent years. Multiresolution algorithms such as Discrete Wavelet Transforms (DWT), Finite Ridgelet Transform (FRIT) and Curvelet Transform (CT) are highly suitable for a number of image processing applications such as detection of line singularities and contiguous edges, edge detection, image compression, video compression and surveillance [38, 39], image denoising, image segmentation, medical imaging [40], astronomical imaging [41, 42], High Definition (HDTV) [43] and digital cinema [44, 45].

Recently, the ridgelet and curvelet transforms [46–48] have been generating a lot of interest due to their superior performance over wavelets. The wavelet transform has been extensively used in image and video processing during the last ten years. However, it has long been known that the wavelet transform has many limitations when it comes to representing straight lines and edges in image processing. While wavelets have been very successful in applications such as denoising and compact approximations of images containing zero dimensional (point) singularities, they do not isolate the smoothness along edges that occur in images. Wavelets are thus more appropriate for the reconstruction of sharp point-like singularities than lines or edges. These shortcomings of wavelets are well addressed by the ridgelet and curvelet transforms, as they extend the functionality of wavelets to higher dimensional singularities, and are effective tools to perform sparse directional analysis. The applications for these algorithms (specially medical imaging) are computationally intensive, which require real-time acceleration process and make hardware acceleration essential.

1.1 Hardware Acceleration

Manipulating large data sets and execution of complex algorithms such as multiresolution analysis make the appearance of hardware acceleration inevitable for providing the necessary performance. Hardware design paradigms are also undergoing a sea change in order to address the various issues involved. There is a perceptible shift towards top-down design flow and a preference for modular, integrated and flexible solutions.

Currently, there are range of processors such as General Purpose Processors (GPPs), Dig-

ital Signal Processors (DSPs), Application Specific Integrated Circuits (ASICs), Graphics Processing Unit (GPU) and Field Programable Gate Arrays (FPGAs) which are used as hardware platforms to resolve complex and intensive applications. These processors have their own preference and flexibility. For instance, GPPs and DSPs, in consequence of the overhead paid for the flexibility, processors are rather inefficient regarding performance and power consumption [49]. ASICs are efficient regarding performance and power consumption, but they lack flexibility, as no programmable resources are provided [49]. The realisation of reconfigurable systems was enabled through the introduction of the FPGAs in the mid eighties. FPGAs share with ASICs the capability to implement application-specific circuits, with the key difference that FPGA circuits are programmed by means of a configuration data stream that specifies the logical functionality and connectivity [49, 50]. FPGAs can perform mathematical operation on an entire vector or matrix at the same time. FPGAs have been evolving and improving rapidly, and have consistently shown the fastest rates of performance gains and supports the fine-grained parallelism of many pipelined DSP applications. The flexibility and performance of FPGA with other rivals are illustrated in Fig. 1.1.

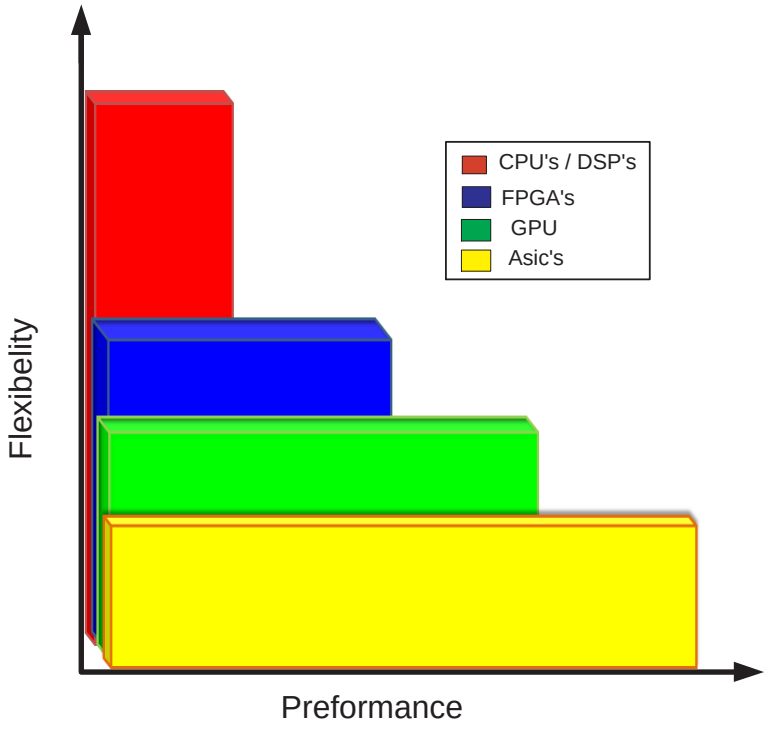


Figure 1.1: Flexibility and performance of FPGA with other rivals [1]

The quality of different implementation approaches can be evaluated using various metrics

such as performance, energy consumption, logic area, cost, time to market and flexibility to incorporate functional requirement changes. These issues recently become the top agenda for venders and designers considerations.

Table 1.1: The quality comparison of different hardware implementation approaches [1]

	Performance	**Cost**	**Power**	**Flexibility**	**Design Cycle**	**Time to Market**
ASIC	High	High	Low	Low	High	High
DSP	Medium	Medium	Medium	Medium	Medium	Mediam
GPP	Low	Low	Medium	High	Low	Low
GPU	High	Medium	High	Medium	Medium	Medium
FPGA	Medium/High	Low	Medium	High	Low	low

In the past two decades, performance and cost have become more significant as digital image processing has migrated from predominantly military and scientific applications into numerous low-cost consumer applications. Energy consumption has also become an important measure as digital image processing techniques have been widely applied in portable, battery-operated systems and devices. Design cycle and time to market also playing an important role as there is always a high demand in the market. Finally, flexibility has emerged as one of the key differentiators in image processing implementations since it allows changes to system functionality at various points in the design life cycle. These trade-offs have resulted in the five primary implementation options which are indicated in Table 1.1. Each implementation option presents different trade-offs in terms of performance, cost, power and flexibility [1]. The quality comparison of different hardware implementation approaches is shown in Table 1.1. A lot of research has been carried out into several areas of architectural support for complex applications using different hardware approaches, some of the conventional approaches for hardware acceleration are listed as follow:

1.1.1 Application Specific Integrated Circuits Hardware

ASICs are designed specifically to perform a given computation and consequently they efficiently perform the given computation according to the application's design objectives. ASICs have some advantages and drawbacks compare to other rivals. They support large, complex design with high performance and low power consumption. Meantime, they are

difficult to design, have long development time and after fabrication the circuit can not be altered. This forces a re-design and a re-fabrication of any part of the chip which requires modification. This is an expensive process, especially when one consider the difficulties in replacing ASICs in a large deployed system [51, 52].
The main disadvantages of this approach can be summarised in the three following points:

- Special purpose hardware has a long development time, from design through simulation and fabrication;
- It can also be expensive if it is a one-off solution or if the volume required cannot justify its fabrication costs; and
- Once this special purpose hardware is built, it is not possible to change the hardware to accommodate slightly different needs. With such a solution a new piece of hardware is usually required for each new algorithm.

The underlying concept behind structured ASICs is fairly simple, it contains a small amount of generic logic implemented either as gates and/or multiplexers and/or a Look-Up Table (LUT). Depending on the particular architecture, the tile may contain one or more registers and possibly a very small amount of local Random Access Memory (RAM). An array (sea) of these tiles is then prefabricated across the face of the chip. Structured ASICs also typically contain additional prefabricated elements, which may include configurable general-purpose Input/Output (I/O), microprocessor cores, gigabit transceivers, embedded (block) RAM and so forth. Structured ASIC technology is especially suitable for platform ASIC designs that have integrated most of the Intellectual Property (IP) blocks and leave some space for custom changes [53, 54]. There are different type of ASICs but the following types of ASICs are currently often mentioned:

- Standard cell based ASIC, it is probably the most generic type of ASIC nowadays. This type of ASIC is constructed from the cell library usually provided by the semiconductor fabrication plant. The set of standard cells is automatically synthesized from the RTL description on VHDL or Verilog;
- Full-custom ASIC, it differs from the previous type of ASIC in that the ASIC designer designs some logic cells and/or layout specifically for this ASIC. Some of the cells can still be taken from the library. This approach has to be used when the

standard library doesn't contain some specific cells or when the standard cells don't meet some specific requirements on performance, area or power dissipation; and

- Structured ASIC, it is quite a different term that shouldn't be confused with the previous ones. This term is usually used in connection with ASICs built on basis of FPGA. Major FPGA vendors can produce chips pre-configured with customer's FPGA bitstream, for the price a few times lower than that of FPGA. The design costs are thereby similar to the design costs for FPGA projects. This can be a cost-effective solution for medium-volume products. This technique is implemented by Xilinx under the brand EasyPath and by Altera under the brand HardCopy

A more recent approach, which aims to benefit from the advantages of special purpose hardware by avoiding many of its disadvantages, is to use dynamically reprogrammable hardware in the form of FPGAs. A brief overview of FPGAs is provided in the next section.

1.1.2 Digital Signal Processors

DSP processors are microprocessors designed to perform digital signal processing the mathematical manipulation of digitally represented signals. DSP is one of the core technologies in rapidly growing application areas such as wireless communications, audio and video processing, and industrial control. DSP was first introduction of the first commercially successful DSP chips in the early 1980s and have been used intensively in last two decades for signal processing applications. General-purpose microprocessors and operating systems can execute DSP algorithms successfully, but are not suitable for use in portable devices such as mobile phones and Personal Digital Assistant (PDAs) because of power supply and space constraints. A specialised DSP will tend to provide a lower-cost solution, with better performance, lower latency, and no requirements for specialised cooling or large batteries. DSP processor has features designed to support high-performance, repetitive and numerically intensive tasks. Some of the DSP features are described as follow [55]:

- The ability to perform one or more multiply-accumulate operations (often called MACs) in a single instruction cycle. The multiply-accumulate operation is useful in DSP algorithms that involve computing a vector dot product, such as digital filters, correlation, and Fourier transforms. To achieve a single-cycle MAC, DSP processors integrate multiply-accumulate hardware into the main data path of the processor;

- Most DSPs are able to complete several accesses to memory in a single instruction cycle. This allows the processor to fetch an instruction while simultaneously fetching operands and/or storing the result of a previous instruction to memory. For example, in calculating the vector dot product for an FIR filter, most DSP processors are able to perform a MAC while simultaneously loading the data sample and coefficient for the next MAC. Such single cycle multiple memory accesses are often subject to many restrictions;

- Specialised execution control. Usually, DSP processors provide a loop instruction that allows tight loops to be repeated without spending any instruction cycles for updating and testing the loop counter or for jumping back to the top of the loop;

- DSPs have one or more dedicated address generation units to speed arithmetic processing. Once the appropriate addressing registers have been configured, the address generation unit operates in the background (i.e., without using the main data path of the processor), forming the addresses required for operand accesses in parallel with the execution of arithmetic instructions. In contrast, general-purpose processors often require extra cycles to generate the addresses needed to load operands;

- DSP processors are known for their irregular instruction sets, which generally allow several operations to be encoded in a single instruction. For example, a processor that uses 32-bit instructions may encode two additions, two multiplications and four 16-bit data moves into a single instruction. In general, DSP processor instruction sets allow a data move to be performed in parallel with an arithmetic operation. GPPs, in contrast, usually specify a single operation per instruction;

- DSPs have low-cost, high-performance input and output, most DSP processors incorporate one or more serial or parallel I/O interfaces, and specialized I/O handling mechanisms such as low-overhead interrupts and direct memory access (DMA) to allow data transfers to proceed with little or no intervention from the rest of the processor.

As a result, DSPs have been successfully used in a wide range of image processing applications [56, 57]. They provide the computing power necessary to process large amounts of data in real-time [58].

1.1.3 Graphic Processing Unit

GPU is a specialised microprocessor that off-loads and accelerates 3D or 2D graphics rendering from the microprocessor. It is used in embedded systems, mobile phones, personal computers, workstations, and game consoles. Modern GPUs are very efficient at manipulating computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for a range of complex algorithms. In a personal computer, a GPU can be present on a video card, or it can be on the motherboard. More than 90% of new desktop and notebook computers have integrated GPUs, but these GPUs are not as power full as those ones which come on a dedicated video card for intensive applications. There are many companies have produced GPUs under a number of brand names, but currently, Intel, NVIDIA and AMD/ATI are the market leaders.

A GPU is a processor attached to a graphics card dedicated to calculating floating point operations. A graphics accelerator incorporates custom microchips which contain special mathematical operations commonly used in graphics rendering. The efficiency of the microchips therefore determines the effectiveness of the graphics accelerator. They are mainly used for playing 3D games or high-end 3D rendering. A GPU implements a number of graphics primitive operations in a way that makes running them much faster than drawing directly to the screen with the host CPU. Modern GPUs also have support for 3D computer graphics, and typically include digital video related functions.

1.2 Field Programmable Gate Arrays

FPGAs are first introduced in the mid 1980 and they have steadily established themselves as an alternative for implementing digital logic in systems. FPGAs were originally created to serve as a hybrid device between Programmable Arrays Logic (PALs) and Mask Programmable Gate Arrays (MPGAs). First generation FPGAs were used to provide a designer solution for glue logic, but now they have expanded their applications to the point that it is not uncommon to find FPGAs as the central processing devices within systems. However, the flexibility, capacity, and performance of these devices have opened up completely new avenues in high-performance computation, forming the basis of reconfigurable computing [59]

Nowadays FPGAs can be found everywhere. Researchers and students use them in to run experiments. Companies use them on development boards to help refine new chip designs.

Companies and universities are using them in cutting-edge research on topics ranging from programming technology, cryptography to real-time systems. As a result of rapid advances in the semiconductor industry, FPGAs themselves are getting so inexpensive that some companies do not even fabricate an ASIC. They simply include the FPGA in their final product. The considerable interest in reconfigurable hardware has been highlighted by an increasing amount of research carried out in the area, coupled with the development of several commercial systems based on FPGAs. There is no doubt that this level of interest will certainly continue to grow over the next number of years. With the emergence of such reconfigurable hardware it is not surprising that there has been wide ranging research into the use of FPGAs to increase the performance of a wide range of computationally intensive applications.

1.2.1 FPGA Structure

The basic architecture of FPGAs consists of three kinds of components logic blocks, routing and I/O blocks. Generally, FPGAs consist of an array of programmable logic blocks that can be interconnected to each other as well as to the programmable I/O blocks through some sort of programmable routing architecture. Fig. 1.2 provides a very simplified diagram of a generic FPGA architecture.

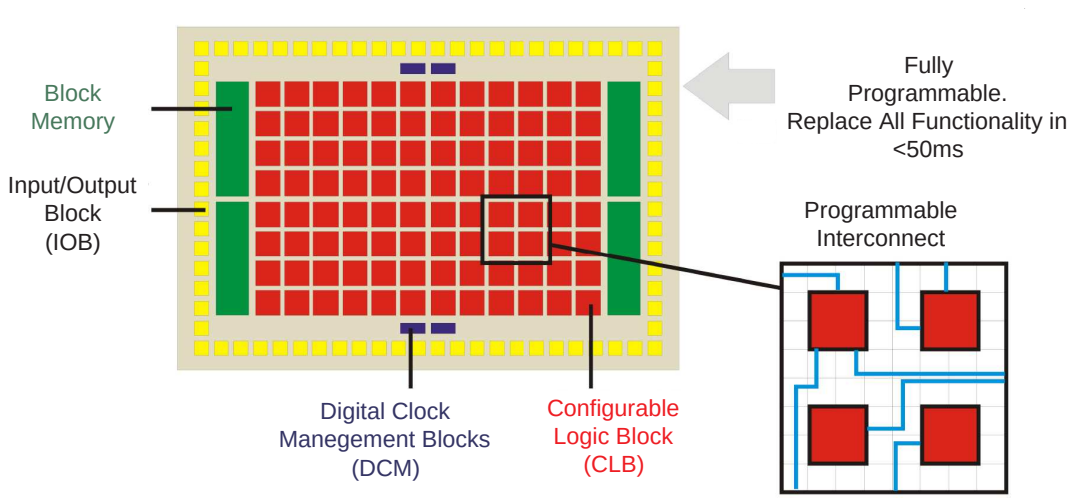


Figure 1.2: Basic elements of the FPGA architecture [2]

Configurable Logic Block

The Configurable Logic Blocks (CLBs) are the main logic resources for implementing sequential as well as combinatorial circuits. Each CLB element is connected to a switch matrix for access to the general routing matrix. Each CLB contains number of slices which depends on FPGA, some FPGA CLB contains two slices and some contains four slices. For instance Virtex-5 CLB captains two slices and each slices contains four LUTs. These slices do not have direct connections to each other, and each slice is organised as a column. Each slice in a column has an independent carry chain. For each CLB, slices in the bottom of the CLB are labeled as slice(0), and slices in the top of the CLB are labeled as slice(1). A typical Virtex-5 CLB block architecture is illustrate in Fig. 1.3.

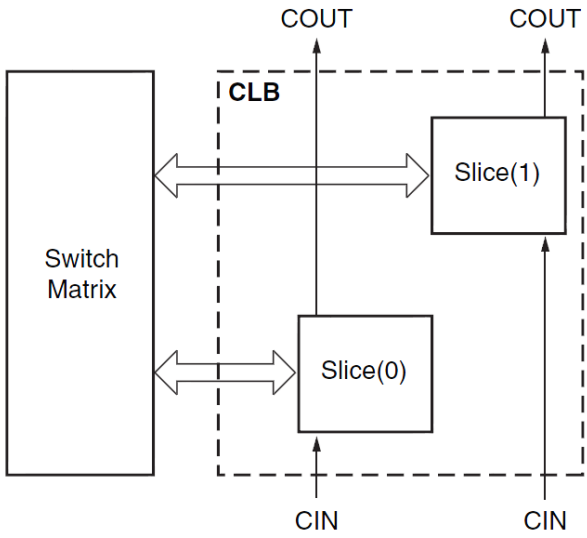


Figure 1.3: Configurable Logic Block Architecture for Virtex-5 FPGA [3]

Slices

A typical advanced slice contains four 6-input LUTs, four Data-Flip-Flops (DFF) and multiplexers [3]. The LUTs are implemented as true 6-LUTs, rather than being constructed using smaller LUTs that can be optionally combined together via multiplexer. The true 6-LUTs provide excellent performance for implementing large 6-input logic functions. Slices receive/produce carry signals from/to neighboring slices for implementing fast arithmetic functions. The LUTs allow any function to be implemented, providing generic logic. The DFF can be used for pipelining, registers, state holding functions for finite state machines, or any other situation where clocking is required. The fast carry logic is a special resource

provided in the cell to speed up carry-based computations, such as addition, parity, wide AND operations, and other functions. The multiplexers are used to combine up to four function generators to provide any function of seven or eight inputs in a slice. As there has been a great deal of experimentation in FPGA logic block architectures, there has been equally as much investigation into interconnect structures [59]. A typical advanced Xilinx Virtex-5 FPGA slice is shown in Fig. 1.4.

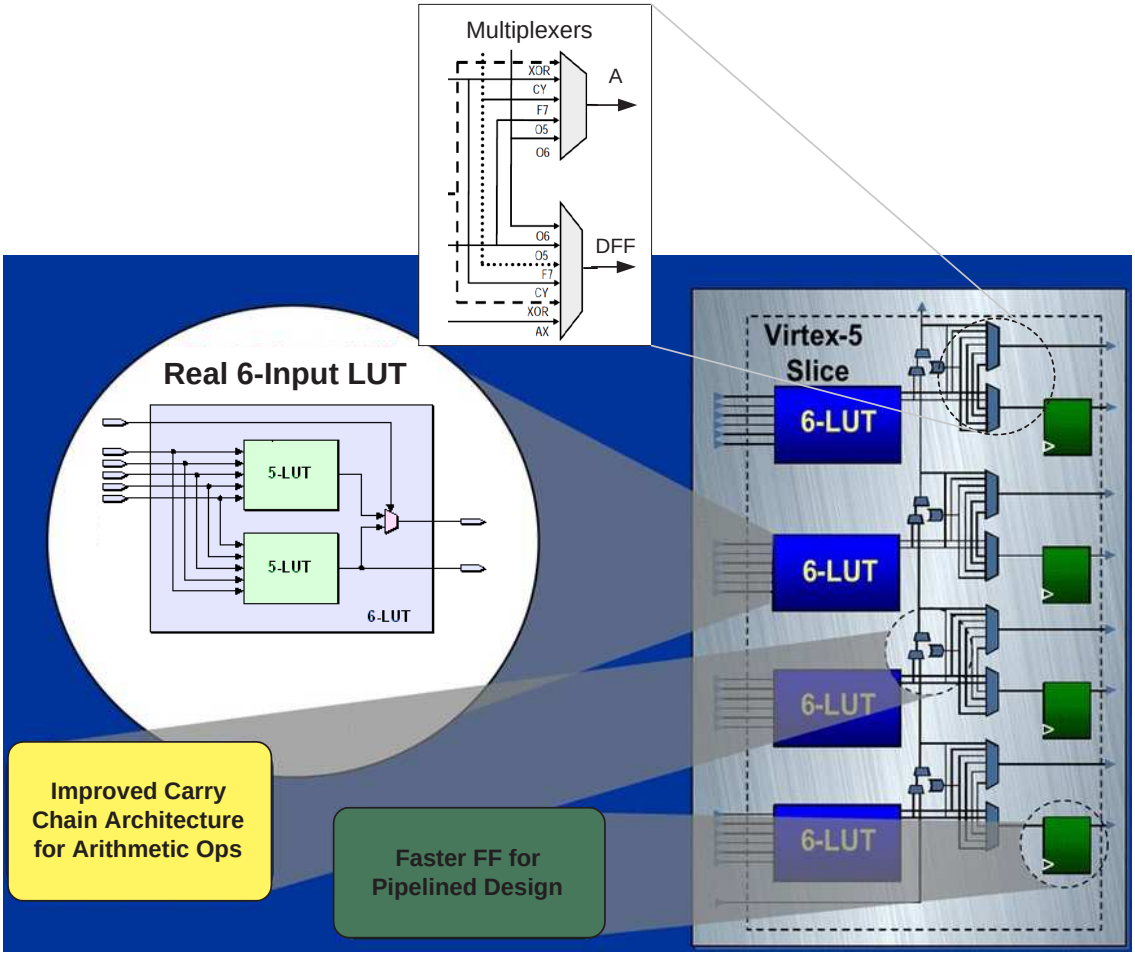


Figure 1.4: Virtex-5 FPGA slice architecture [4]

Routing

The routing architecture is designed to handle versatile connection configurations. Most FPGA architectures organise their routing structures as a relatively smooth sea of routing resources, allowing fast and efficient communication along the rows and columns of logic blocks. The logic blocks are embedded in a general routing structure, with input and output signals attaching to the routing fabric through connection blocks [59].

Connection Blocks

The connection blocks provide programmable multiplexers, selecting which of the signals in the given routing channel will be connected to the logic block's terminals. These blocks also connect shorter local wires to longer distance routing resources. Signals flow from the logic block into the connection block and then along longer wires within the routing channels [59] the typical connecting block is also shown in Fig. 1.2.

Switch Boxes

At the switch boxes there are connections between the horizontal and vertical routing resources to allow signals to change their routing direction. Once the signal has traversed through routing resources and intervening switch boxes, it arrives at the destination logic block through one of its local connection blocks. In this manner, relatively arbitrary interconnections can be achieved between the logic blocks in the system. While the routing architecture of an FPGA is typically quite complex - the connection blocks and switch boxes surrounding a single logic block typically have thousands of programming points - they are designed to be able to support fairly arbitrary interconnection patterns [59]. Fig. 1.5 shows a typical switch box architecture for FPGA.

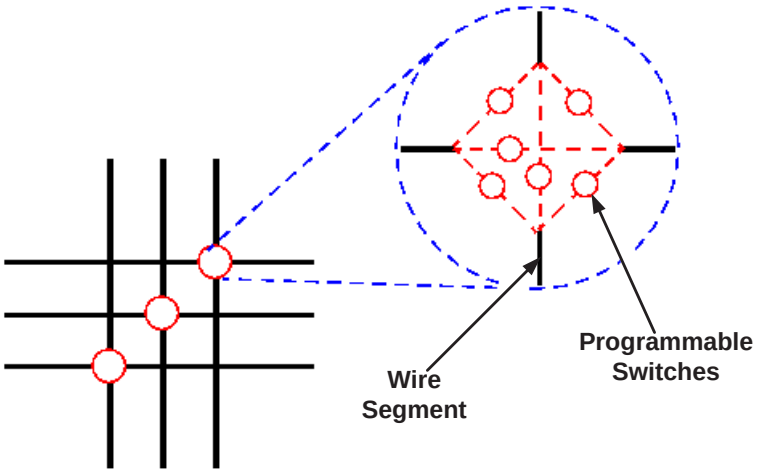


Figure 1.5: FPGA switch box [5]

1.2.2 FPGA Design Flow and Synthesis

A typical design flow for FPGA design is given in Fig. 1.6. It consists of a number of tools: high-level design languages (Handel C which has been used for most implementations

carried out in this research project), Hardware Description Languages (HDLs), schematic capture tools, simulation tools, netlist converters and Place And Route (PAR) tools.

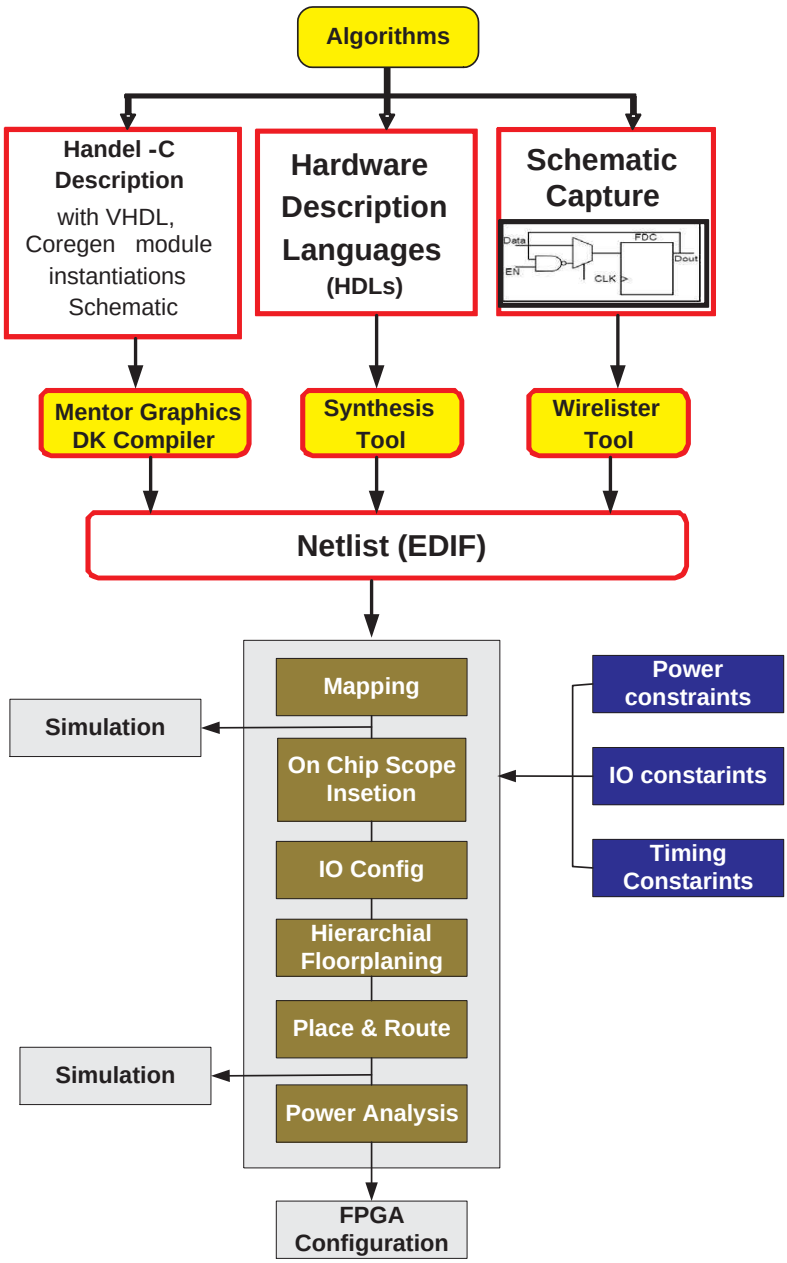


Figure 1.6: FPGA design cycle

Schematic Design Entry

Schematic tools provide a graphic interface for design entry. You can use these tools to connect symbols representing the logic components in your design. You can build your design with individual gates, or you can combine gates to create functional blocks. Schematic

capture tools allow the developer to specify a circuit at a low level as a 2D diagram by connecting together logic components. This can be done using one of a number of front end design tools (such as Xilinx ISE, Viewlogic's ViewDraw and etc [2, 60]). The logic components are usually contained in a library supplied by the FPGA vendor. Because of the vendor specific nature of the components, schematic designs produced for a specific FPGA architecture are not easily portable to other architectures. Once a design has been specified using schematic capture, it can be converted into a netlist by a schematic-to-netlist converter tool, such as Viewlogic's Wirelister [61]. The Schematic Editor, Symbol Editor, and Library Manager allows to capture the users design with a combination of schematics, FPGA library macros, or HDL modules. Hierarchy is established by instantiating a symbol from a customer symbol produced by the IPexpress or your own RTL module. The major schematic tools are listed below:

- Schematic editor, a graphical editor for placing library symbols, adding attributes, and wiring connections
- Symbol editor,a graphical editor for creating and modifying library symbols
- Library manager, a graphical interface to collect library symbols
- Hierarchy navigator, a graphical tool to explore schematic/HDL design hierarchy

Hardware Description Languages (HDLs)

Hardware description language or HDL is a computer language or programming language for formal description of electronic circuits, and more specifically, digital logic. It can describe the circuit's operation, its design and organization, and tests to verify its operation by means of simulation. HDLs are standard text-based expressions of the spatial and temporal structure and behaviour of electronic systems. Like concurrent programming languages, HDL syntax and semantics includes explicit notations for expressing concurrency. However, in contrast to most software programming languages, HDLs also include an explicit notion of time, which is a primary attribute of hardware. Languages whose only characteristic is to express circuit connectivity between a hierarchy of blocks are properly classified as netlist languages used on electric computer-aided design (CAD). The two most commonly used are Very High Speed Integrated Circuits Hardware Description Language (VHDL) [62, 63] and Verilog [64].

Handel-C Language

Handel-C is a high level programming language which targets low-level hardware, most commonly used in the programming of FPGAs. It is a rich subset of C, with non-standard extensions to control hardware instantiation with an emphasis on parallelism. Unlike many other design languages that target a specific architecture Handel-C can be compiled to a number of design languages and then synthesised to the corresponding hardware. This frees developers to concentrate on the programming task at hand rather than the idiosyncrasies of a specific design language and architecture. Handel-C allows hardware to be directly targeted from software, allowing a more efficient implementation to be created. In [65–68] have shown that Handel-C shortens design time by a factor of 3-4 times with approximately the same operating speed compared to traditional HDLs. A number of recent projects developed under Handel-C illustrate the languages wide applications fit.

- Internet Security, Data Encryption Standard (DES) encryption algorithm in hardware for Secure Sockets Layer (SSL) acceleration.
- Digital Music, MP3 decoding in reconfigurable hardware.
- Internet Telephony, voice-over-IP phone implementing H.323 and Transmission Control Protocol (TCP)/IP in hardware.
- Image Processing, accelerating complex image processing algorithms in FPGAs.

Synthesis and Netlist Representation

Design synthesis can be defined as the transformation of a design to a level of lower abstraction. This definition is sometimes refined to the transformation of a design from a point in the functional domain to one in the structural domain. CAD tools can be used to perform synthesis tasks at different design points and with various levels of interactivity with the designer. Logic synthesis and high-level synthesis as two of the major synthesis areas used in digital design. High Level Synthesis (HLS) is the translation of an algorithmic design specification into an interconnection of combinational logic (functional units) and the Register Transfer Level (RTL). The actual logic descriptions at the RTL are specified in a generic manner perhaps as boolean equations or as generic gates. It is the task of logic synthesis to map these descriptions onto a specific structure suitable for the target architecture. After running synthesis the designs become netlist files that are

accepted as input to the next implementation steps. Netlist describes the connectivity of an electronic design and usually convey connectivity information and provide nothing more than instances, nets, and perhaps some attributes. Most netlists either contain the descriptions of the parts or devices used in the design. Each time a part is used in a netlist, this is called an instance and each instance has a master or definition. These definitions will usually list the connections that can be made to that kind of device, and some basic properties of that device. These connection points are called ports or pins, among several other names [2]. The netlist format can be in the standard Electronic Data Interchange Format (EDIF) format [69], or in another vendor specific format (e.g. Xilinx Netlist Format -XNF from Xilinx).

Simulation

simulation is used to verify the functionality of the design early in the design flow by simulating the HDL description. Testing the design through the simulation before the design is implemented at the RTL or gate level allows user to make any necessary changes early in the design process. There are normally two way of simulating the design effectively. One, with larger hierarchical HDL designs, perform separate simulations on each module before testing the entire design. This makes it easier to debug your code. Second, create a test bench to verify that the entire design functions as planned. Use the same test bench again for the final timing simulation to confirm that the design works as expected under worst-case delay conditions. The Xilinx tools are normally used to simulate are ISE simulator and ModelSim.

Place And Route

Place and Route Tools (PAR) is a map stage in the design of integrated circuits, and FPGA. As implied by the name, it is composed of two steps, placement and routing. The first step, placement, involves deciding where to place all electronic components, circuitry, and logic elements in a generally limited amount of space. This is followed by routing, which decides the exact design of all the wires needed to connect the placed components. This step must implement all the desired connections while following the rules and limitations of the manufacturing process. Routing between the cells is then performed based on this placement and the routing resources available. Timing-driven routing is automatically invoked if PAR finds timing constraints associated with the design

(e.g. in a user constraint file). After PAR operation a bitstream file can be generated which can be used to configure the FPGA [61].

1.3 The Importance of Power in FGPA

Recent advances in semiconductor process technology has led to rapid scaling of transistor dimensions, allowing a large number of them to be packed on the same chip, this make power dissipation a very important issue and a very high attention is required in the early stages of the design cycle. Poor design choices early on in the design cycle can result in expensive corrections and modifications. Taking this issues into account, the advent of battery operated devices and increased deployment of processing in energy and thermal constrained environments such as satellites has accelerated interest in power awareness as a key requirement of the design process.

1.3.1 Motivations for Power Awareness

The well known Moore's Law [70] has been the guiding beacon for the electronics industry. He stated that the number of transistors that can be placed inexpensively on an integrated circuit has doubled approximately every two years. The trend has continued for more than half a century and is not expected to stop until 2015 or later. This is entirely true for FPGAs as well. FPGA vendors are embracing latest cutting edge fabrication technologies resulting in a quadrupling of FPGA capabilities every three years and latest FPGAs have even surpassed the one billion transistor mark [2]. The Moore's law diagram which shows the increase of transistors and power is presented in Fig. 1.7. The logic growth of FPGAs with relation to Moore's law is shown in Fig. 1.8. The growth of power density with computation is shown in Fig. 1.9

Based on information provided in Fig. 1.7, 1.8, 1.9 and the implementation of power hungry algorithms on FPGAs, it can be concluded that there is a huge demand for adoption of power aware design practices for FPGAs. The motivations for lowering power and power aware design are listed below:

- Advancement in electronics continuously bring products with bitter resolution and better color display such as mobile computing, camera and gaming functions and high-speed communication. processing these applications become more complicated

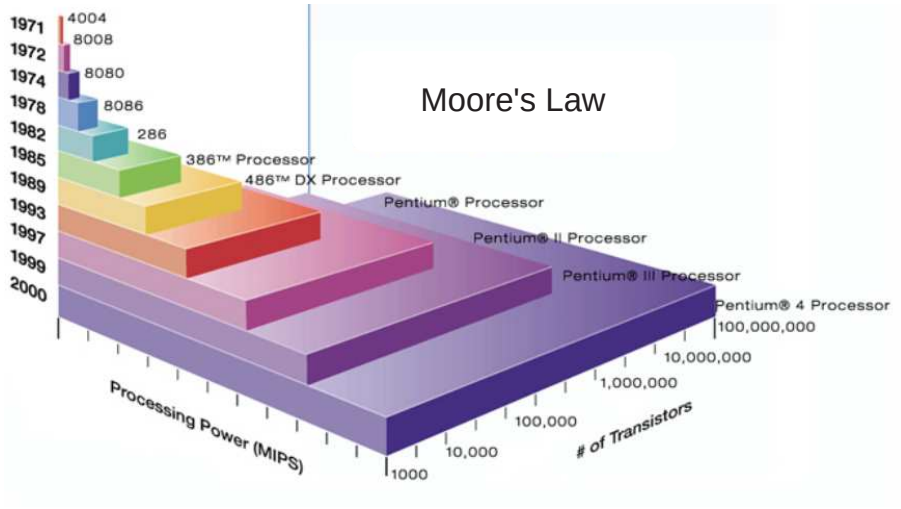


Figure 1.7: G. Moore's law which shows the increase of transistors every year [6]

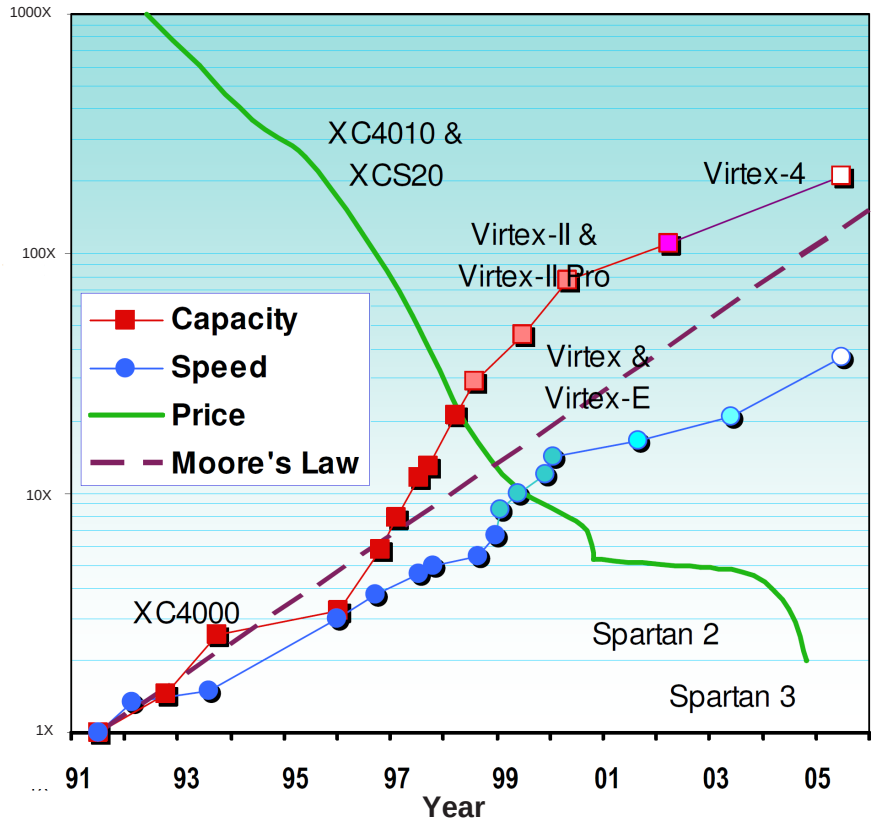


Figure 1.8: Growth of FPGA with relation to Moore's law [2]

they consume more power, which drives the need for a power aware design to improve the efficiency and reduce the size of new generation batteries;

- The mobility is an important issue in a number of environments and portable devices, a power aware design will help reduce the size of battery and consume less power.

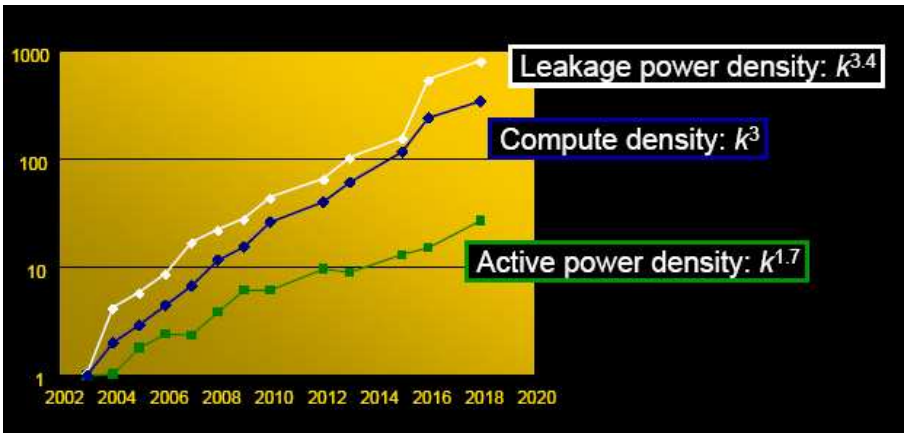


Figure 1.9: Power density comparison with computation [2]

- Thermal stability and noise immunity are the most important issues to be acknowledged during the design cycle. A computer owner in Britain demonstrated the extent of the thermal and heat density issues by placing a dish of aluminum foil above the chip inside his PC and frying an egg for breakfast [71]. The Passive components, such as resistors, capacitors and inductors typically change in value with temperature unpredictable operation, leakage currents typically increase. Some applications such as satellites and network chips are very susceptible to such effects, due to lower tolerance and high sensitivity to the effects of drift. Interference can be unacceptable in sensitive environments such as satellites and Magnetic Resonance Imaging (MRI) scanners where FPGAs are used heavily for processing large volumes of data.

- Environmental concerns is also on of the main factor for lowering the power due to global warming.

1.3.2 FPGA Power Dissipation Details

Compared to other custom chips, FPGAs contain long routing tracks with significant parasitic capacitance. During high-speed operations, the switching activity on these long routing tracks causes significant power dissipation. Power dissipation calculations for FPGAs are similar to other complementary metal-oxide semiconductor application-specific integrated circuit (Complementary Metal Oxide Semiconductor (CMOS) ASIC) devices. The total power for FPGA can be broken into two categories Static Power (SP) and Dynamic Power (DP). DP is consumed due to the switching of gates and is still responsible for a large percentage of the total power dissipated in current computing devices, although

power dissipation related to SP is expected to increase in the future. The DP consumption of FPGAs can be separated into three main part: data-path, synchronisation and off-chip power which is shown in Fig. 1.10.

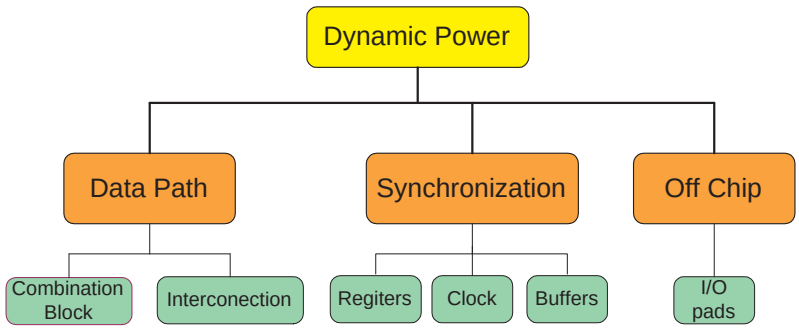


Figure 1.10: FPGAs dynamic power dissipation

The FPGAs DP is classified into clock, signal, logic, input and output power. Power consumption of the data-path interconnection (programmability) is the highest of the three parts and increases linearly with the input clocking frequency. Various techniques (including pipelining and partitioning, clock gating, bus multiplexing, asynchronous design and clock frequency reduction) can be applied to an FPGA design to reduce this power consumption.

Importance of Power Modelling

Power dissipation is becoming a major concern for semiconductor vendors and customers. If current design trends continue, a typical microprocessor will consume 50 times more power than that can be supported by cost-effective packaging techniques. It is clear that power will become one of the two most serious design concerns (along with design complexity) in coming process generations. FPGAs will not escape this trend; already, FPGA vendors report that power consumption is one of the primary concerns of their customers. Compared to ASICs and other custom chips, FPGAs contain long routing tracks with significant parasitic capacitance; during high speed operations, the switching activity on these long routing tracks causes significant power dissipation.

There have also been numerous CAD algorithms that target low power. Often, these studies rely primarily on reducing switching activity to result in a low-power solution. Although reducing switching activity does lower the power, power also depends on the architecture, the lengths of critical signal routes, the rise and fall times of the signals, and the amount of static power. Though neglected in the past, static power is expected to

become an increasingly important part of the total power. In order to adequately evaluate these new CAD algorithms and techniques, designers are working hard to optomise their designs in way to reduce the power dissipation as minimum as possible.

One way to overcome the issue of power dissipation is the development of power modelling tools that provide reliable estimates of the power, to enable the designer to make the right design choices while optimising the IP core. Models have always been important for electronic system design at all levels, whether at the component (Integrated Circuit (IC) design), board or system level. lately, large FPGAs can easily exceed 5-10W power dissipation as a function of the foregoing variables. Currently, most approaches to hardware power estimation and modelling operate at the Register-Transfer Level (RTL) or lower levels of design abstraction. Attempts at power estimation for functional descriptions have suffered from poor accuracy because the design decisions performed during their synthesis lead to an unavoidable, large uncertainty in any power estimate that is based solely on the functional description.

1.4 Research Objectives and Motivations

Computationally intensive image applications are power hungry and require a large amount of memory which causes a high demand for efficient algorithms implementation to overcome the problems. Most of all, the processing time is a real concern which lead to high demand for acceleration. The development of advanced and fast reconfigurable hardware in form of FPGAs bring a huge interest in real-time image processing which can perform mathematical operation on an entire vector or matrix at the same time. FPGAs are emerging to be one of the most reliable platform for intensive image and signal processing algorithm due to its high performance, security, low design-turnaround time, low power and reconfigurability. FPGAs are everywhere companies use them on development boards to help refine new chip designs. Students use them in the classroom to run experiments. Companies and universities are using them in cutting edge research on topics ranging from programming technology to cryptography to real time systems. The parts themselves are getting so inexpensive that some companies do not even fabricate an ASIC, they simply include the FPGA in their final product.

There is no doubt that this level of interest will certainly continue to grow over the next number of years. It is not surprising that there has been a considerable amount of research

into the use of FPGAs to increase the performance of a wide range of computationally intensive applications. However, wider acceptance of FPGAs as a replacement to traditional hardware and software platforms for performance enhancement acceleration depends on the ability of FPGA based designers to learn from experiences in the ASIC design domain and to evolve solutions to the greatest FPGA design challenges for the next decade adopting higher level design paradigms and simultaneously addressing the challenges of power consumption. Keeping these challenges in perspective, the key objectives of this research project can be broadly summarised as follows:

- To develop novel architectures for MAAs using advanced arithmetic techniques and design methodologies through the optimisation strategies at various abstraction levels;

  - To explore and implement efficiently HWT on FPGAs using factorisation methodologies for a range of 1-D 2D transforms;

  - To implement the proposed FRIT architecture efficiently on FPGAs using pipelining and parallelism techniques for a range of 1-D and 2D transforms;

  - To develop and implement cyclic convolution on FPGAs using parallelism and systolisation which can be integrated in mutiresulation architectures as the main building block for efficient computation; and

- To further evaluate the accuracy of a high level power modelling for proposed custom IP cores such as HWT FRIT and etc.

1.5 Overall Project Plan

Image and Vision Systems (IVS) research lab at Brunel university, led by Dr A. Amira has developed a number of high performance cores that have been grouped together into a library. It is the aim of this work to explore the library and add a couple of multiresolution algorithms IP cores to existing library. The existing cores (prior to the commencement of the work in this thesis) can be broadly classified into image and signal processing transforms and other matrix operations including decompositions. The library includes transform based cores such as the Finite Ridgelet Transform (FRIT) [14], Fast Hadamard Transform (FHT) [11,72], Discrete Hartley Transform (DHT) [73], Fast Fourier Transform

(FFT) [74], Discrete Wavelet Transform (DWT) [75], etc. Matrix operations based cores in this library include matrix multiplication and Colour Space Conversion (CSC) [76]. Decomposition related cores that have been implemented include Singular Value Decomposition (SVD). Historically, computational efficiency and compact area footprint were the key objectives in the design of the cores in this library. This research work is primarily concerned with optimising multiresolution analysis cores the from library with particular emphasis on applying techniques for minimising power and energy consumption. The other objective of this work is to design and implement additional novel and optimised cores to be added to the library. It is worth mentioning that this research is also focusing on further evaluation of a high level power modelling in terms of accuracy for the proposed custom multiresolution analysis cores.

1.6 Organisation of the Thesis

The structure of this thesis is as follows. Chapter 2 takes a closer look at the most recent architectures and systems for the various IP cores that have been developed in this research work including HWT, FRAT, FRIT and cyclic convolution. A detailed and thorough of various power modelling approaches for FPGAs are also provided in this chapter. High performance and power efficient architectures of HWT based on Distributed Arithmetic (DA) principles and sparse matrix factorisation techniques, which is suitable for FPGA implementation are presented in Chapter 3. Architectural techniques such as parallelism, and pipelining have been exploited to yield efficient and power aware architectures for FRAT and FRIT are discussed in Chapter 4. Chapter 5 explores optimised architectures and efficient FPGA implementation for cyclic convolution using systolisation, parallelism and pipelining. In Chapter 6, further evaluation of high level power modelling methodology and mathematical formulation of this model which has been applied to number of benchmark circuits developed in the previous chapters are presented. Concluding remarks and opportunities for future work are presented in Chapter 7.

Chapter 2

Literature Review

2.1 Introduction

In recent years, Multiresolution Analysis Algorithms (MAAs) have proven to be useful decomposition tools in a wide variety of applications throughout mathematics, science, and engineering. For example, image compression standard known as JPEG2000 and video compression standard, MPEG-4, are entirely wavelet based. In addition, MAAs have been used in many image processing applications such as image segmentation (medical imaging), edge detection for object recognition etc. Among other MAAs Discrete Wavelet Transform (DWT), Haar Wavelet Transform (HWT) and some latest transforms including Finite Ridgelet Transform (FRIT) which is a combination of Finite Radon Transfrom (FRAT) and HWT and Curvelet transforms are gaining the momentum to be used for these applications. Among other image processing applications medical imaging is one of the hottest topic in the field. There is a high demand for an accurate processin and acceleration. The nature of medical images defers form other images and have some features (involving lots curves, line and edge discontinuity) which are very important to be considered during the process. These applications are matrix based transform and data intensive which involve large operation, power hungry and computability intensive [77]. Researchers are working round the clock to improve the performance of matrix multiplication algorithms. One way of obtaining high performance for these algorithms is through implementation of hardware acceleration which has described in details in chapter 1.

This chapter takes a closer look at the most recent architectures for MAAs (FRIT and

HWT) and cyclic convolution which are implemented on different reconfigurable hardware. A number of performance metrics including area occupied, maximum frequency, throughput rate, power dissipation and so on need to be considered while validating the work presented in this thesis against comparable existing work. In the area of power modelling, the nature of comparison needs to include quantitative and qualitative measures such as model accuracy, parametrisation, scalability and model abstraction level among others. In order to benchmark our IP cores and modelling methodology with the best in class, a thorough and extensive literature survey has been conducted and updated throughout the period of research in this work.

The rest of this chapter is organised as follows. Existing architectures and FPGA implementations for FRAT, FRIT, HWT and cyclic convolution are presented in Section 2.2. An overview of the existing power modelling methodologies for FPGAs is presented in Section 2.3. A synopsis of the shortcomings of existing work and concluding remarks are provided in Sections 2.4 and 2.5 respectively.

2.2 FPGA Implementations of Selected Algorithms and Related Architectures

In this section, the existing architectures and implementations for a number of matrix operation based algorithms including HWT, FRAT, FRIT and cyclic convolution are presented.

2.2.1 Existing Architectures and FPGA Implementation of Wavelet Transform Filters

In the past two decades, there has been an ever increasing amount of interest in wavelet transform. Wavelet transform is an emerging signal processing technique that can be used to represent real life non-stationary signals with high efficiency. Efficient hardware implementation and acceleration of the algorithms used in these applications are becoming very important due to the amount of time required for its processing and the real need for embedded solutions in most of the advanced image and vision systems. In the following sections a number of existing architectures for wavelet transform filters and their hardware implementation are presented.

Acceleration of Haar Wavelet Transform Using FPGA

In [7] an approach to accelerate the Haar-classifier based on face detection algorithm with highly pipelined micro-architecture and parallel arithmetic is presented. The proposed FPGA architecture is presented in Fig. 2.1. It is also reported that their approach is flexible toward the resources available on the FPGA chip. This work provides an understanding toward using FPGA for implementing non-systolic based vision algorithm acceleration. The proposed approach has been implemented on a HiTech Global PCIe card that contains a Xilinx Vitex-5 (XC5VLX110T) FPGA chip. The integral image stores into the BRAM (Block RAM) as a 32×32 array. During pixel scan mode, in each cycle, FPGA reads out one line of the integral image to the 17×17 buffer sub-window, and latches the buffer sub-window data to the 17×17 image window every 17 cycles as the source images integral sub-window to compute with the classifiers.

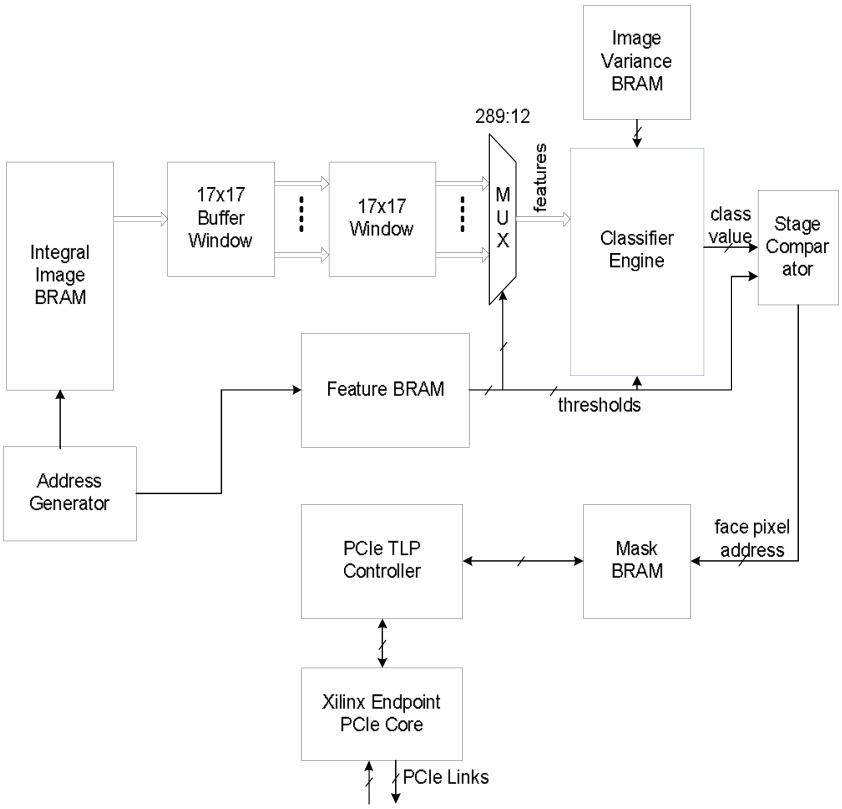


Figure 2.1: FPGA block diagram [7]

In this work it has been reported that the software version, the Haar classifier face detection application could only achieve performance of 5 frames/sec, while for 1-classifier FPGA implementation 37 frames/sec, and 16-classifier FPGA implementation 98 frames/sec.

Wavelet Transform Implementation based on Distributed Arithmetic

In [8] a parallel based Distributed Arithmetic (DA) FPGA implementation of the Discrete Wavelet Transform (DWT) and its inverse has been described. In [8] they use the maximal utilization of the look-up table architecture of the FPGAs platform by reformulating the wavelet computation in accordance with the parallel DA algorithm. In this work, a DA implementation of the Daubechies 8-tap wavelet FIR filter which consists of LUT, shift registers and a scaling accumulator is presented. The LUT stores all possible sums of the Daubechies 8-tap wavelet coefficients.

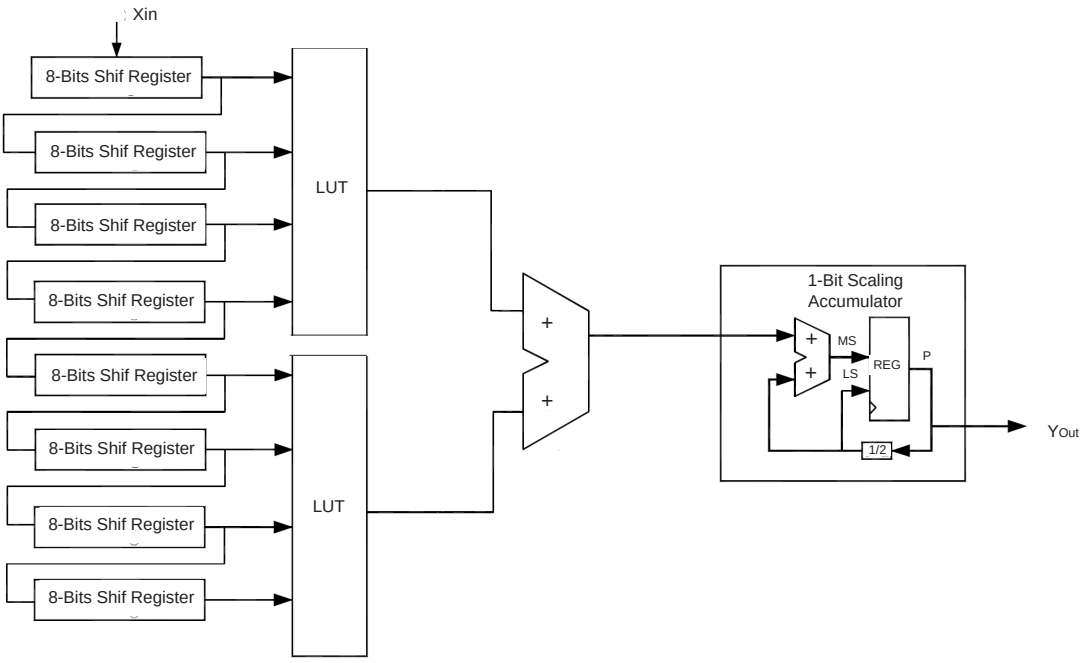


Figure 2.2: A partitioned-LUT DA implementation of the Daubechies FIR filter [8]

Since the LUT size increases exponentially with the number of coefficients, the 8-bit LUT is decomposed into two 4-bit LUTs which called Serial Distributed Arithmetic (SDA) is shown in Fig. 2.2, which corresponds to partitioning the input sample into sub-samples and processing these sub-samples in parallel. The outputs using a two-input accumulator. The 4-bit LUT partitioning is optimum in terms of logic resources utilization, since this matches naturally the Virtex FPGA slice architecture.

A parallel implementation of the inherently serial distributed arithmetic (SDA) FIR filter, shown in Figure 4, corresponds to partitioning the input sample into M sub-samples and processing these sub-samples in parallel.

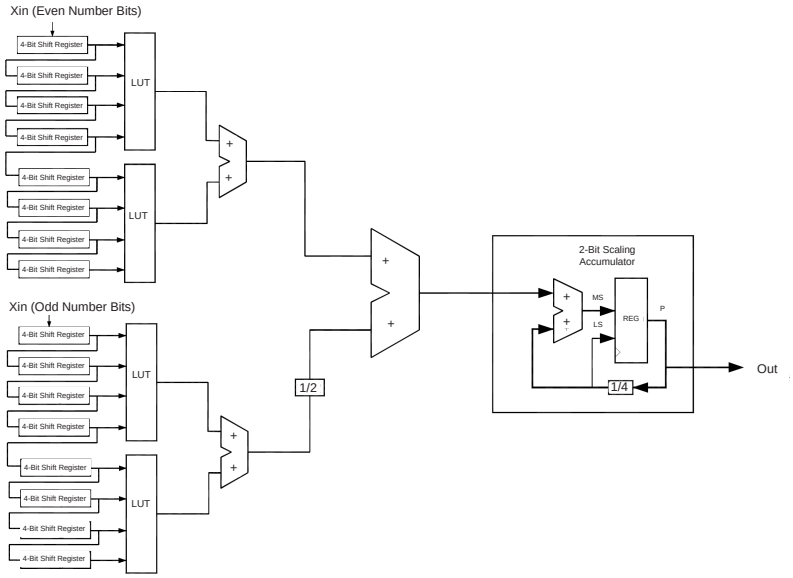


Figure 2.3: A parallel distributed arithmetic Daubechies FIR filter [8]

A 2-bit Parallel Distributed Arithmetic (PDA) FIR filter implementation is shown in Fig. 2.3. It corresponds to feeding the odd bits of the input sample to a SDA LUT adder tree, while feeding the even bits simultaneously to an identical tree. Compared to the SDA filter, shown is Fig. 2.2, each shift register is replaced with two similar shift registers at half the bit size. The odd bit partials are left shifted to properly weight the result and added to the even partials before accumulating the aggregate by a 1-bit scaling adder. Finally, since two bits are taken at a time, the scaling accumulator is changed from 1-to-2-bit shift (1/4) for scaling. The design has been implemented on Virtex XCV300 with slices utilization of 645 and maximum frequency of 48.1 MHz.

FPGA Implementation of Haar Wavelet Transform for Imaging

In [9] an image coding processing scheme based on a variant of HWT which uses only addition and subtraction process is presented. After computing the transform, the selection and coding of the coefficients is performed using a methodology optimized to attain the lowest hardware implementation complexity. Coefficients are sorted in groups according to the number of pixels used in their computing. The proposed architecture of the 2D HWT processor for images is presented in Fig. 2.4. The first block computes the one-dimensional transform using one 1D-transform processor, being the intermediate results stored in a memory. In the 2D-HWT architecture, the first 1D-HWT transform is applied to each row of the complete image (pixels in raster order), the intermediate coefficients are

stored in the intermediate memory, IMem; then, a new 1D-HWT transform is applied to each column of IMem to obtain the 2D-HWT of the complete image. The 1D-processors use only two adder/subtracters and a register file to store the intermediate data.

In order to verify the operation of the designed architecture in this work, an implementation has been made using a mixed type VHDL/schematic description. Images with 256x256 pixels (256 grey levels) have been used. The devices used to verify its architecture is the ALTERA APEX20K FPGA.

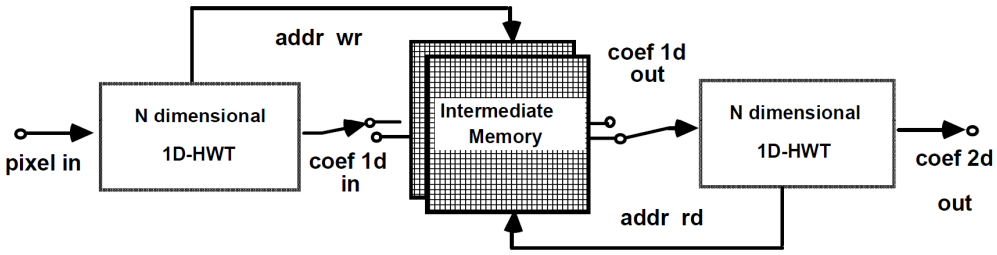


Figure 2.4: Architecture for the computation of the 2D-HWT based on the 1D-HWT [9]

It uses only one general-purpose configurable EP20K60ETC144-1 of the ALTERA APEX20K family. The core processor structure, the 2D-WHT, is of the 1D-HWT Intermediate Memory 1D-HWT type. This intermediate memory (dual-port RAM type) is external to the chip, its size being that corresponding to two images. The encoder processor uses only 97.2% of the FPGA Logic Elements (LEs), 128 ESB and 96of the available pins.

Efficient Reconfigurable Architecture for Wavelet Transform

In [10] a unified computation framework for DWT and Continuous Wavelet Transform (CWT) based on lifting scheme is presented. A reconfigurable architecture that includes reconfigurable lifting step arrays and reconfigurable address generator is also presented. In order to validate this architecture, an FPGA prototype is built to test the reconfiguration of 2D discrete 5/3 and 9/7 transforms (defined in JPEG2000) and 2D HWT. The proposed reconfigurable architecture for DWT and CWT is present in Fig. 2.5.

Reconfigurable Address Generator (RAG) is responsible for calculating the address of access to the two dual-port SRAM memories. Reconfigurable Lifting Step Array (RLSA) is connected by some reconfigurable lifting step kernels. RLSA can be configured for different wavelet transforms by modifying some parameters such as the number of delay registers and pipeline registers. The two dual-port SRAM memories are used for storage of

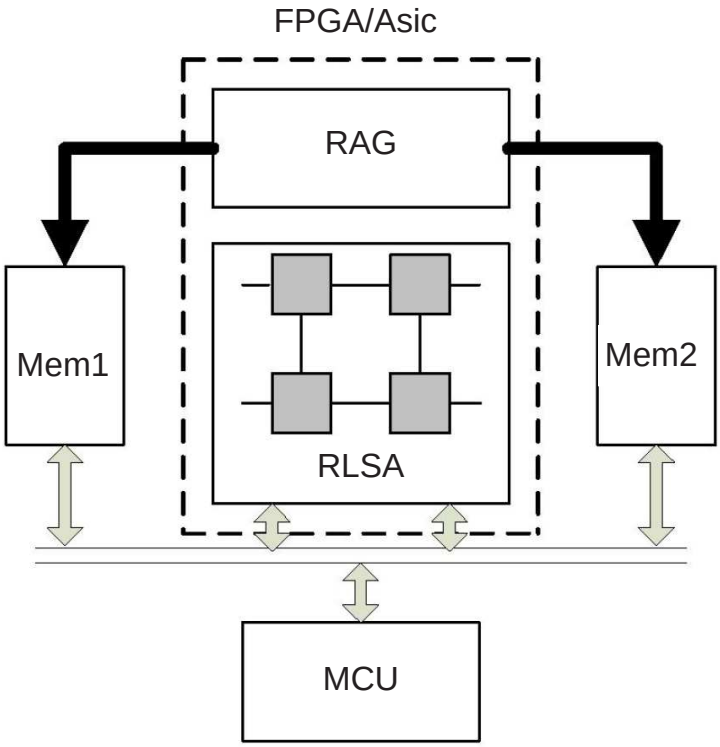


Figure 2.5: Reconfigurable architecture for DWT and CWT [10]

image or signal source and computation results. Multipoint Control Unit (MCU) is used as mean controller. In this work, the 5/3 and 9/7 wavelet filters are chosen for 2D DWT. The HWT which is widely used in image edge detection is taken as the wavelet filter of 2D CWT. The proposed designed has been implemented with frequency of 20 MHz on Xilinx Virtex II (x2v500) FPGA, it utilized 1175 slices, 612 flip flops and 2176 LUTs.

Efficient Architecture for Discrete Biorthogonal Wavelet Transform

In this work, a high-speed/high-throughput architecture for 1-D Discrete Biorthogonal Wavelet Transform (DBWT) is presented. This architecture performs 1-D DBWT decomposition of an N_0 sample input signal with K decomposition levels in $N_0/2$ clock cycles. The architecture offers efficient hardware utilisation for VLSI implementation by combining the linear phase property of biorthogonal filters with decimation. In order to avoid this underutilization, this propose an architecture consisting of two PEs. PE1 performs the first level of decomposition ($k = 1$) where k is the decomposition level, while the second PE2 is responsible for the higher level of decompositions ($2 \leqslant k \leqslant K$) based on Recursive Pyramid Algorithm (RPA) approach. A top-level scheme of the proposed architecture is shown in Fig. 2.6, where the coefficients d and a are representing the detail and approxi-

mation components. It is worth mentioning that this kind of pipelined approach leads to a high-speed /high-throughput, low-power and also highly efficient architecture.

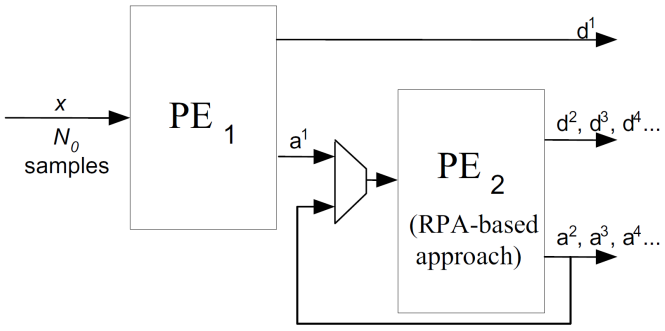


Figure 2.6: Top-level architecture for DBWT [10]

2.2.2 Existing Spare Matrix Factorisation Methodologies

In this section, a brief description about existing and related Spare Matrix Factorisation Methodologies based on different algorithms have been presented.

Sparse Matrix Distributed Arithmetic Based Architecture

In [11] two approaches suitable for FPGA implementation of Fast Hadamard Transforms (FHT) is presented. Two architectures for FHT using both a systolic architecture and distributed arithmetic techniques are also presented in this paper. The first approach uses the Baugh-Wooley multiplication algorithm for a systolic architecture implementation. The second approach is based on both a distributed arithmetic ROM and accumulator structure, and a sparse matrix factorisation technique. To reduce the number of arithmetic operations and to speed up the process, the symmetry in the FHT matrix coefficients and a sparse matrix factorisation are exploited in this architecture. The architecture for transform length $N = 8$ is presented in in this work. Four separate ROM-Accumulate blocks calculate the eight transforms as follows: a butterfly structure of bit-serial adders and subtractors is used to generate the elements of the input matrix as shown in Fig. 2.7. This architecture has an effective computational complexity of $2n$ where n is the wordlength. This DA based architecture is designed and implemented on the Xilinx XCV1000E FPGA of the Virtex-E family [78]. For $N, n = 8$, it is reported that a maximum frequency of approximately 50 MHz is achieved and the design occupies around 75 FPGA slices.

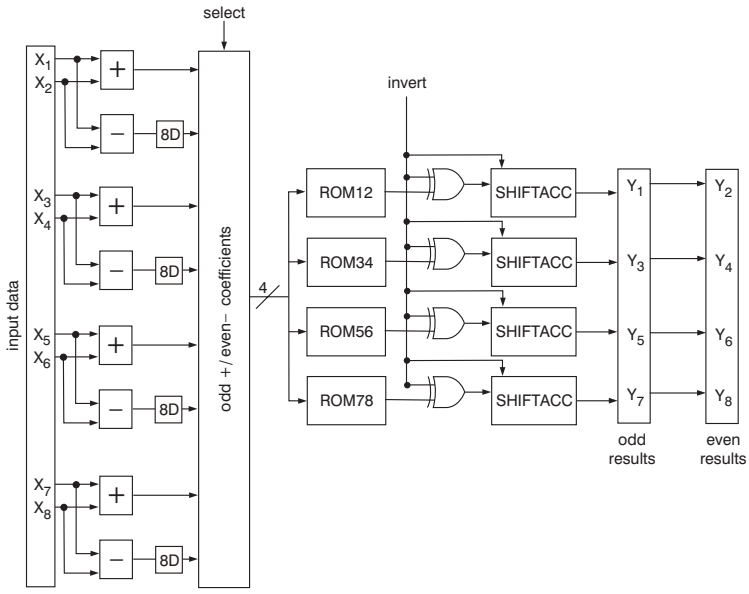


Figure 2.7: DA based architecture for the fast Hadamard transform [11]

Sparse Matrix Factorization Based on DA Architecture

In [12] a novel algorithmic transformation for the FHT based on sparse matrix factorization and DA principles has been presented. The architecture has been parallelized and pipelined in order to achieve high throughput rates. The architecture for the 1-D FHT is shown in Fig. 2.8. The inputs ($W = 8$ bit) are fed in bit-serial fashion from the input port. The ADD/SUB block in the input module operates on every odd clock pulse and their output is appended to the input buffer two words at a time (one addition value and one subtraction value) on every even clock pulse in a systolic manner until the full vector to be transformed has been read.

During the first nine cycles, eight bit-serial outputs for each even vector are produced in parallel and during the second 9 cycles eight bit-serial outputs for each odd vector are produced in parallel. In order to verify the performance of the proposed architecture for FHT, the design has been prototyped on the Celoxica RC1000 [33] board containing the Xilinx XCV2000E FPGA. The implementation results show the design utilised 162 slices with maximum frequency of 115MHz.

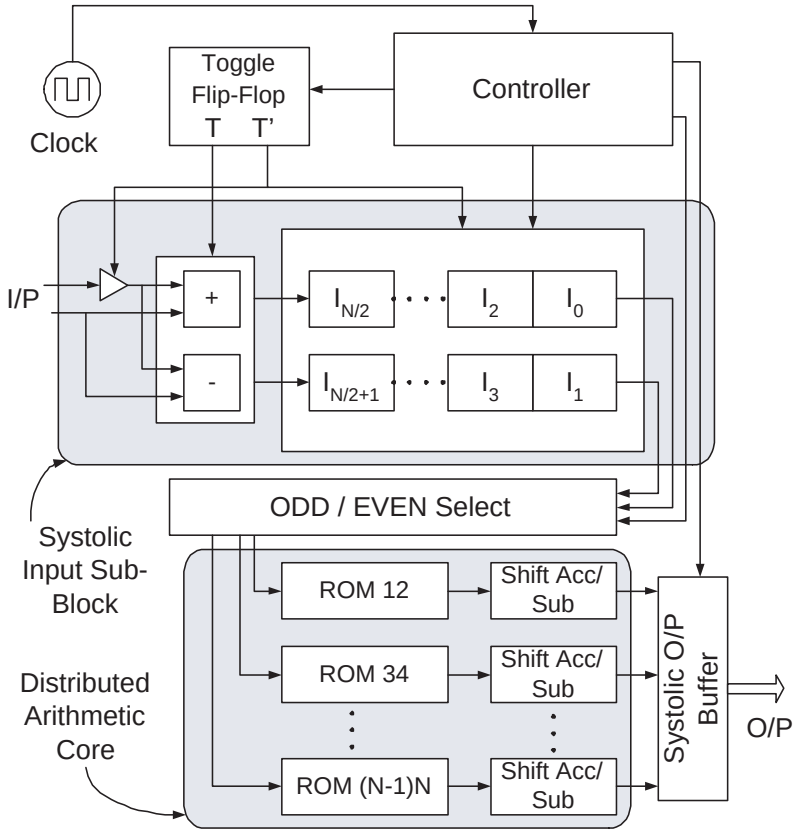


Figure 2.8: DA based architecture for FHT [12]

2.2.3 Existing Architectures and Efficient FPGA Implementation of Finite Radon & Ridgelet Transforms

FRIT has gained attention recently due to superior compaction over wavelets due to its directional nature (different angles projections), resulting in better performance in applications such as compression and denoising. The FRIT is generated by performing a FRAT on a non-dyadic sized block of the source image followed by a wavelet transforms operation. Existing FPGA based implementations of the FRAT and FRIT are presented in this sections.

Serial Inputs Architecture and FPGA Implementation of FRAT

In [13] a serial input architecture for FRAT which is suitable for Ridgelet or Curvelet is presented. The proposed architecture is presented in Fig. 2.9. The input buffer is a linear Distributed RAM with p^2 address locations, where p is the block size. The output buffer is a linear array of shift registers with p locations. The main component of this architecture is the controller. It calculates the list of which points in the input image affect which

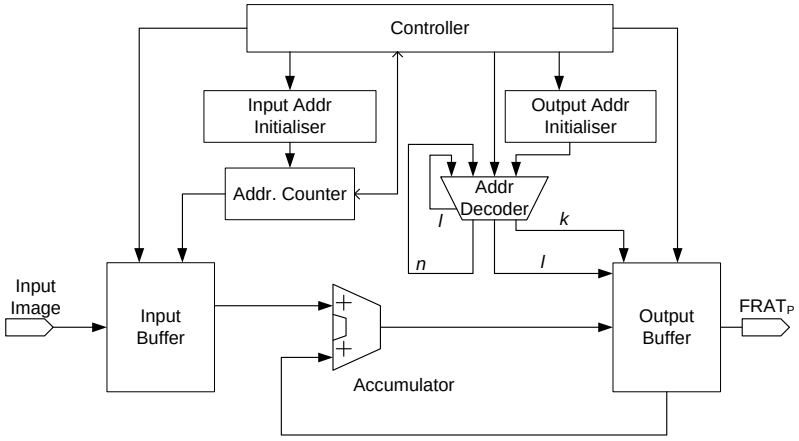


Figure 2.9: Block diagram of proposed FRAT implementation [13]

points in the output image. The accumulator is a L_o-bit accumulator that accumulates the l^{th} pixel value for the k^{th} Radon projection. The input is taken in serial format during the first p^2 clock cycles. The processing takes p^2 clock cycles. The p rows of the FRAT are purged from the output buffer $(p+1)$ times in parallel fashion, once every p^2 clock cycles.

In order to verify the performance of the proposed architecture, the design has been implemented on a Virtex-E2000 FPGA chip with different vector size. The design utilized 198 slices and 112.867 MHz frequency when the block size is 7.

Parallel Cyclic FRAT Architecture

In [14] a parallel, systolic FRAT which is parametrisable, scalable and high performance core with a time complexity of $O(p^2)$, where p is the block size. The FRAT architecture is a serial I/O architecture with a parallel core that computes all $(p+1)$ FRAT vectors simultaneously. The standard pseudocode [79] is not used as a basis for developing the design. Instead, a novel systolic based dereferencing technique is used to compute the FRAT coefficients. The input and core section of the design are completely pipelined. The architecture uses an array of registers that store the address dereferencing values for each FRAT vector. An systolic array is used to store the address dereferencing values. Additionally, instead of using arrays to store the output coefficients, the design choice of using dual ported RAMs helps in reduction of area. The proposed architecture is shown in Fig. 2.10.

The first register in each column is used as the address dereferencer for each output RAM buffer. At the end of p^2 clock cycles, the output buffer contains the image block in the

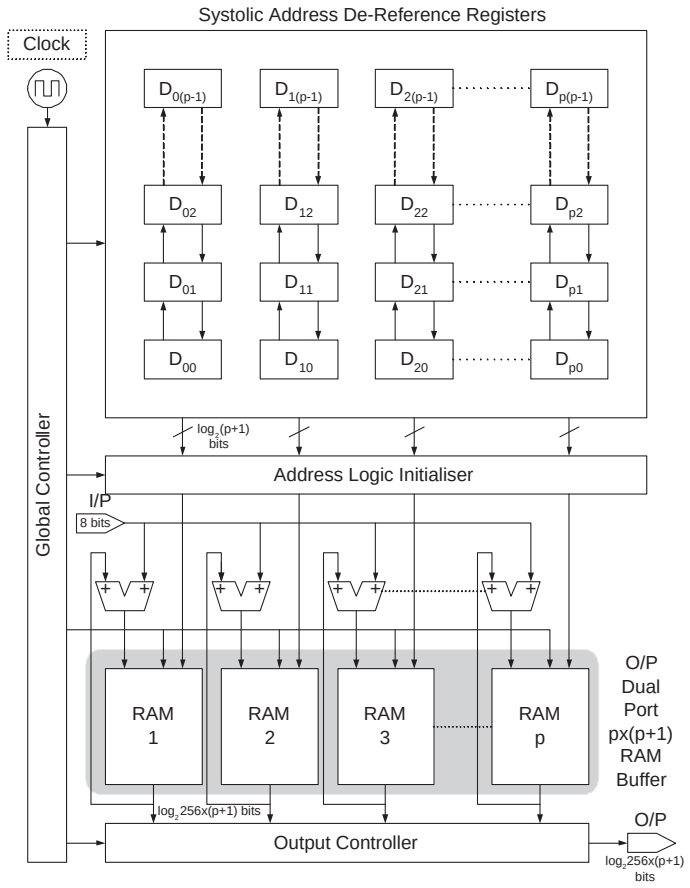


Figure 2.10: FRAT architecture with parallel core [14]

transform domain. They are then ejected in a serial fashion from the output port. In order to verify the performance of the proposed architecture, the design has been prototyped on the Celoxica RC1000 [33] and implemented on Xilinx XC2V8000 (Virtex-II) [80] and XC4VLX200 (Virtex-4) [34] platforms.

Reference and Memoryless Implementation of FRAT

In [15] two architectures for the FRAT and their FPGA implementation have been described. The first architecture is called a reference FRAT architecture and is a direct hardware implementation of a suitable modified variant of the standard FRAT pseudocode [79]. The architecture comprises an address logic initialiser, multiplexer, accumulators and two memory blocks for storing transform vectors. The first proposed archeticture is shown in Fig. 2.11.

The second architecture presented in [15] is a Memoryless FRAT architecture which operates in a parallel manner with p times the throughput of the first architecture, where

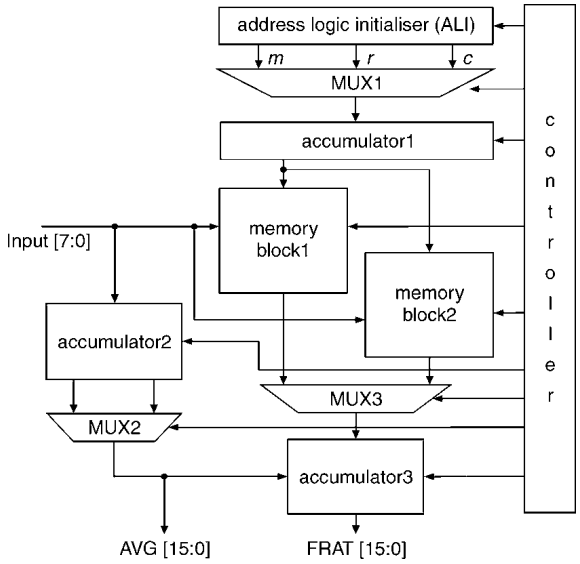


Figure 2.11: Reference architecture for the FRAT [15]

p is the block size. The second presented architecture is shown in Fig. 2.12. Address Logic Initialiser (ALI), wide multiplexer and adder blocks are used as sub-blocks in this architecture. The ALI drives the Address Generators (AG) that turn generate signals to control the address bus. It is worth mentioning that the multiplexer operates on all the p^2 pixels simultaneously in this architecture and hence scalability may not be inherently feasible as a result of wiring complexities.

Both architectures have been designed using Verilog HDL and synthesised by Xilinx ISE development tools using the Virtex-II device family [80]. The reference architecture occupies 159 slices and provides a maximum operating frequency of 100 MHz with a power consumption of 114.98 mW at 50 MHz. The memoryless architecture requires 558 FPGA slices and provides a maximum operating frequency of about 82 MHz. Power is consumed at the rate of 253 mW at 50 MHz. Xilinx XPower [2] is used for power estimation assuming a 1.5 V supply.

Efficient VLSI Architecture and FPGA Implementation of FRIT

In [14] an efficient architecture for the FRIT suitable for VLSI implementation based on a parallel, systolic FRAT and DWT sub-block, respectively is presented. The proposed VLSI architecture with parallel core is shown in Fig. 2.13. The first register in each column is used as the address dereferencer for each output RAM buffer. At the end of p^2 clock cycles, the output buffer contains the image block in the transform domain. They are then

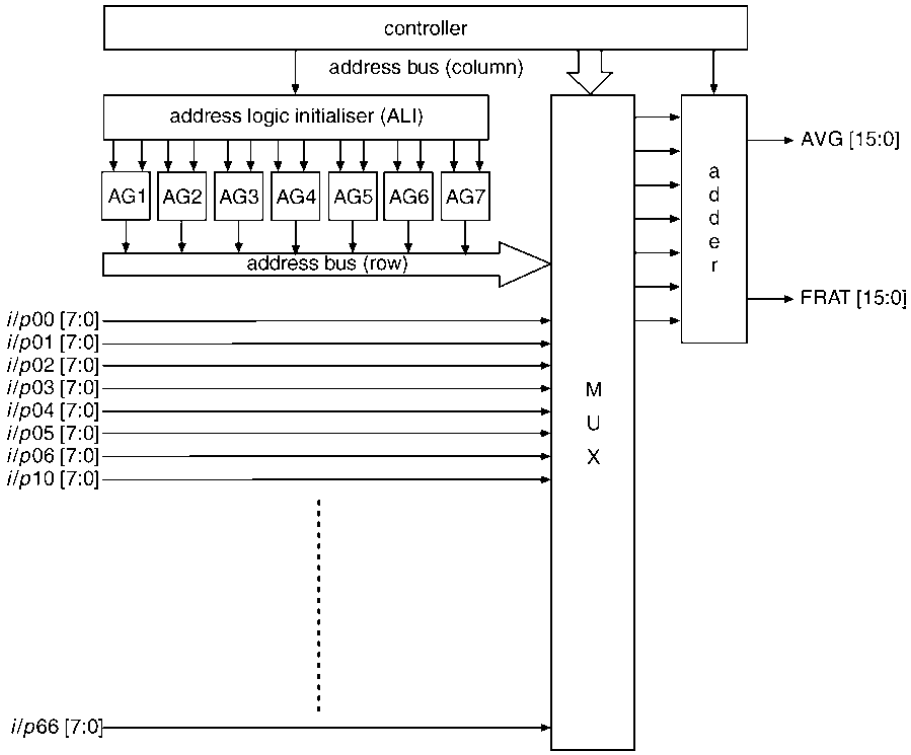


Figure 2.12: Memoryless architecture for the FRAT [15]

ejected in a serial fashion from the output port. In order to verify the performance of the proposed architectures for FRIT, the design has been prototyped on the Celoxica RC1000 board containing the Xilinx XCV2000E FPGA. For a fair comparison with existing work the authors resynthesised and implemented without any architectural modifications for the Xilinx XC2V8000 (Virtex-II) platform. Synthesis is also carried out on the low-cost Spartan 3L platform because it is a popular low cost and low-power FPGA. The FPGA implementation for proposed designs occupies 1,176 slices with 40.60MHz frequency when the block size is 17.

FPGA Implementation of FRIT Based on Standard FRAT and DWT

In [16], a FPGA implementation of the FRIT for image processing applications is presented. The proposed architecture uses the FRAT and 1-D Discrete Biorthogonal Wavelet Transform (DBWT) as building blocks. The architecture of FRAT that is implemented in [16] is a straightforward implementation of the FRAT pseudocode presented in [79] and Appendix B. The address logic initialiser along with controller block constitute the address generator that generates addresses, i.e., $L_{k,l}$ for memory blocks. The accumulator is a L_O-bit accumulator that accumulates the l^{th} pixel value for the k^{th} Radon projec-

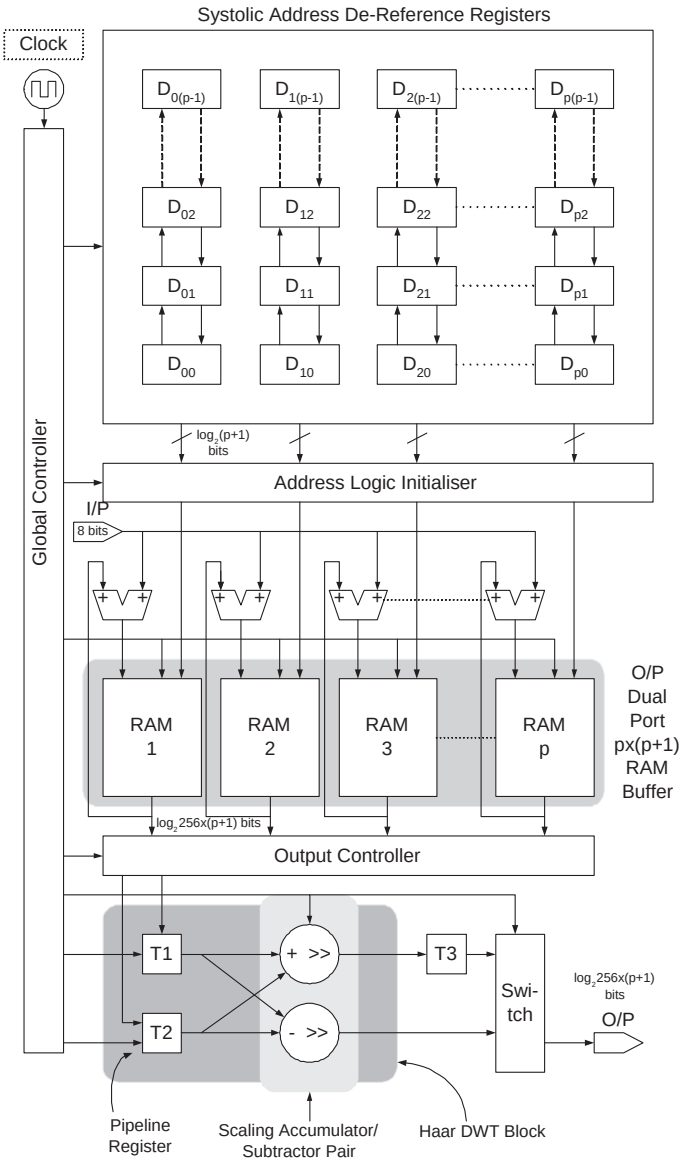


Figure 2.13: FRIT architecture with the FRAT and DWT sub-blocks [14]

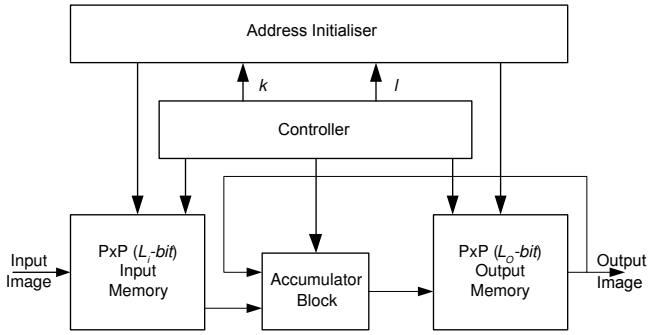


Figure 2.14: Standard pseudocode based architecture for the FRAT [16]

tion. The controller block organises the flow of this process with input and output data flow. Architectural details of the FRAT and the DBWT sub-blocks in [16] are pictorially presented in Figs. 2.14 and 2.15 respectively. In order to verify the performance of these architectures, the designs have been ported to a Xilinx Virtex-II FPGA [80] chip using Handel-C [35]. The implementation results show that the core speed for the FRIT architecture in [16] is around 100MHz and it occupies 491 Slices for an input image size of 7×7 with a distributed RAM based implementation.

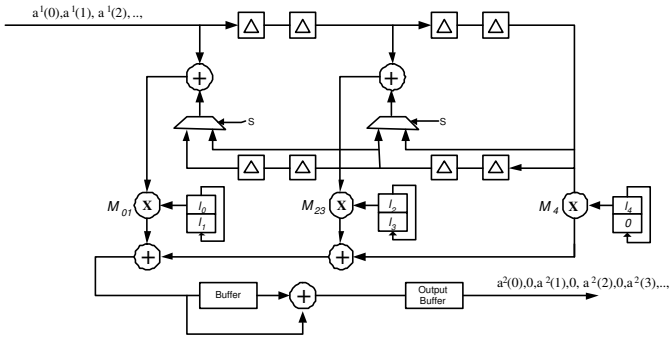


Figure 2.15: DBWT sub-block based on the àtrous algorithm [16]

Generic and Pseudo-code Based Implementation of the FRIT

In [17], two architectures have been presented for the implementation of FRIT. The first architecture uses a generic block that uses a combination of LUTs, matrix of accumulators and multiplexers to perform the FRAT. This is followed by a tree structure based architecture for implementing the DWT. The time complexity of this design is $O(p^4)$, where p is the block size. The maximum frequency for the forward transform using this architecture is reported to be 33MHz. The first proposed architecture for FRAT and DWT is shown in Fig. 2.16 and Fig. 2.17 respectively.

The second FRIT architecture presented in [17] is based on the standard FRAT pseudocode which is also provided in [79] and Appendix B. It has a core time complexity of $O(p^4 \cdot (p+1))$. The Radon Transform Module in this architecture contains a FRAT calculator which uses address generators, accumulators, RAMs and local control logic to perform the FRAT iteratively. Both the Haar and the àtrous algorithm have been implemented for the wavelet sub-section of this FRIT. The proposed FRAT architecture is presented in Fig. 2.18. The block diagram of the wavelet module is reproduced in Fig. 2.19. Each ridgelet block in this architecture consisting one FRAT and one HWT sub-block. The design has been implemented on FPGAs and occupied a total area of 828 slices.

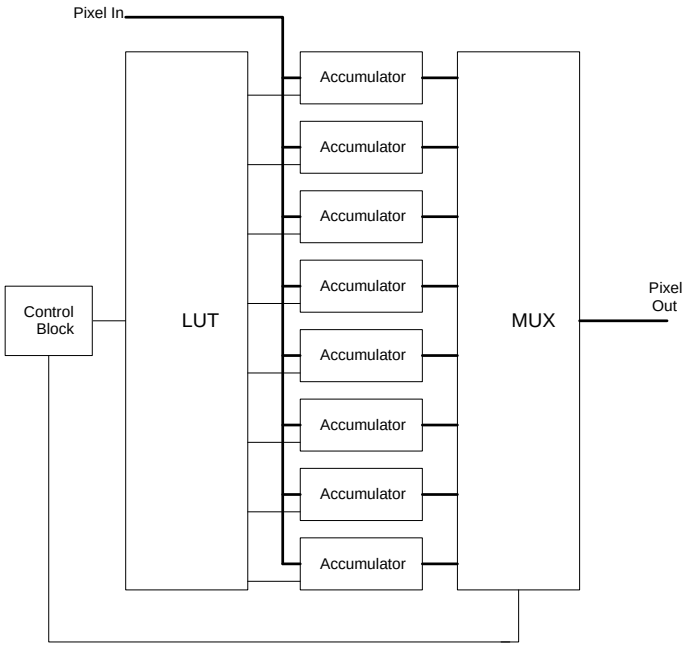


Figure 2.16: Generic architecture for the FRAT [17]

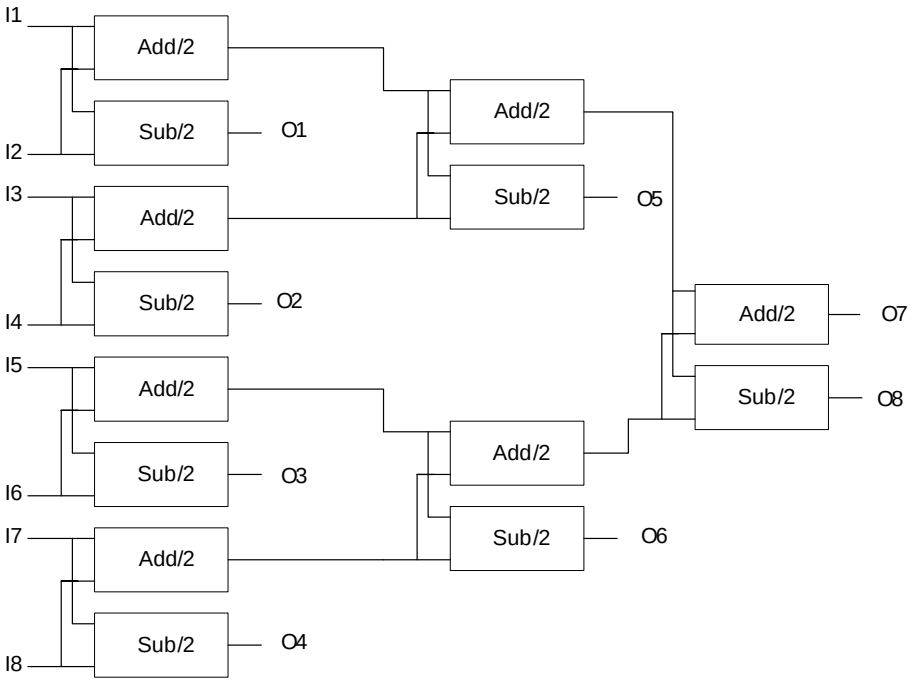


Figure 2.17: Generic tree-based architecture for the DWT [17]

2.2.4 Related Architecture and FPGA implementation of Cyclic Convolution

Many image processing operations such as scaling and rotation require re-sampling or convolution filtering for each pixel in the image. Convolutions on digital images are im-

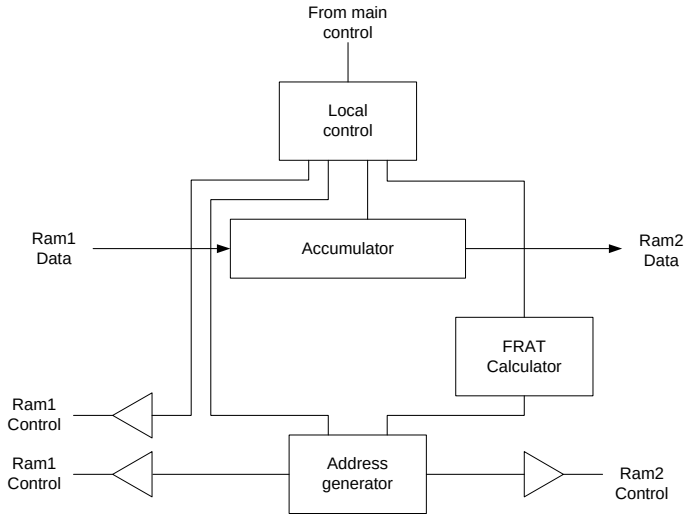


Figure 2.18: Standard pseudocode based architecture for the FRAT [17]

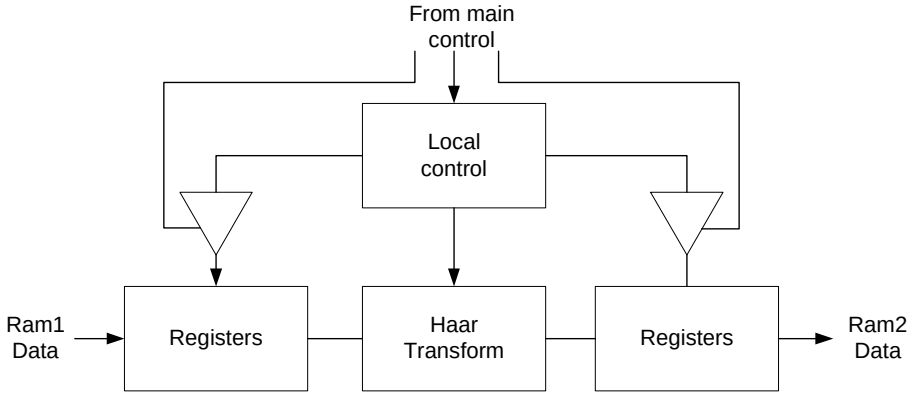


Figure 2.19: DWT sub-block for the standard pseudocode based FRIT architecture [17]

portant since they represent operations that are more general than the operations that can be performed on analog images. Convolution has many applications which have great significance in discrete signal processing. Some of the major uses of convolution are state Image processing; Wavelets generated by using discrete singular convolution kernels and Fourier transform applications. The cyclic convolution of two aperiodic functions occurs when one of them is convolved in the normal way with a periodic summation of the other function. That situation arises in the context of the Circular convolution theorem. The identical operation can also be expressed in terms of the periodic summations of both functions. In this section, the existing architectures and FPGA implementations of cyclic convolution have been presented.

New Systolic Algorithm and Array Architecture of DST Based on Cyclic Convolution

In [18] a new formulation for computing an N-point prime-length Discrete Sine Transform (DST) through two pairs of $[(N-1)/4]$ point cyclic convolutions, where $[(N-1)/4]$ is an odd number is presented. The cyclic convolution based algorithm is used further to obtain a simple regular and locally connected linear systolic array for concurrent pipelined implementation of the DST. The proposed systolic array architecture for cyclic convolution is presented in Fig. 2.20.

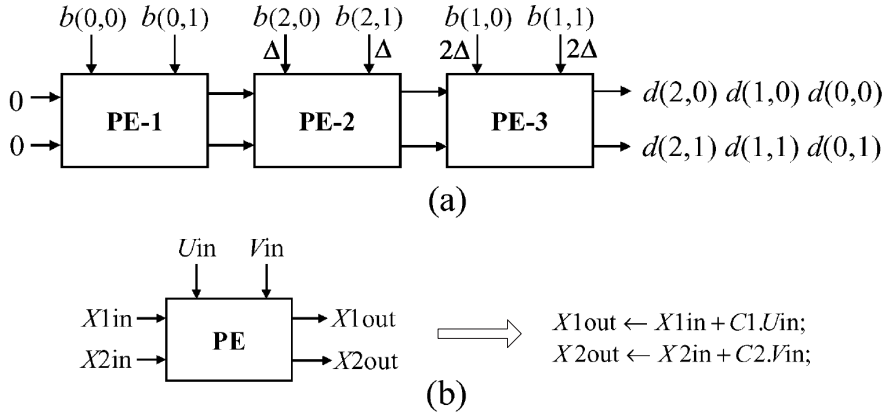


Figure 2.20: Linear systolic array for the computing a pair of three-point cyclic convolutions. (a) The linear array. (b) Function of each PE [18]

The proposed linear systolic array is used to realize a pair of three-point cyclic convolutions, this convolution structure can be used further for the computation of 13-point DST. The input values are fed to the individual PEs through a circularly extended input interface, such that the input values to a PE are staggered by one cycle-period with respect to the preceding PE to maintain the data dependency requirement. The function of each PE of the structure is shown in Fig. 2.20(b) where each PE performs a pair of multiplications $(C1 \times Uin)$ where $C1$ and $C2$ constant coefficient and a pair of additions $(C1 \times Uin)+(X1in)$ in each cycle period. The multiplications in a PE are always performed with a pair of fixed coefficients. This feature of the PEs can be utilized to implement the multiplications in each PE by a pair of Look Up Table (LUT) ROMs that store the product values for all possible input values for the given pair of multiplying coefficients of the PE.

Hardware Efficient Fast DCT Based on Novel Cyclic Convolution Structures

In [19] a new fast cyclic convolution algorithm, which is hardware efficient and suitable for high-speed VLSI implementation, especially when the convolution length is large. The proposed thirteen point DCT hardware implementation using fast cyclic convolution algorithm is presented in Fig. 2.21.

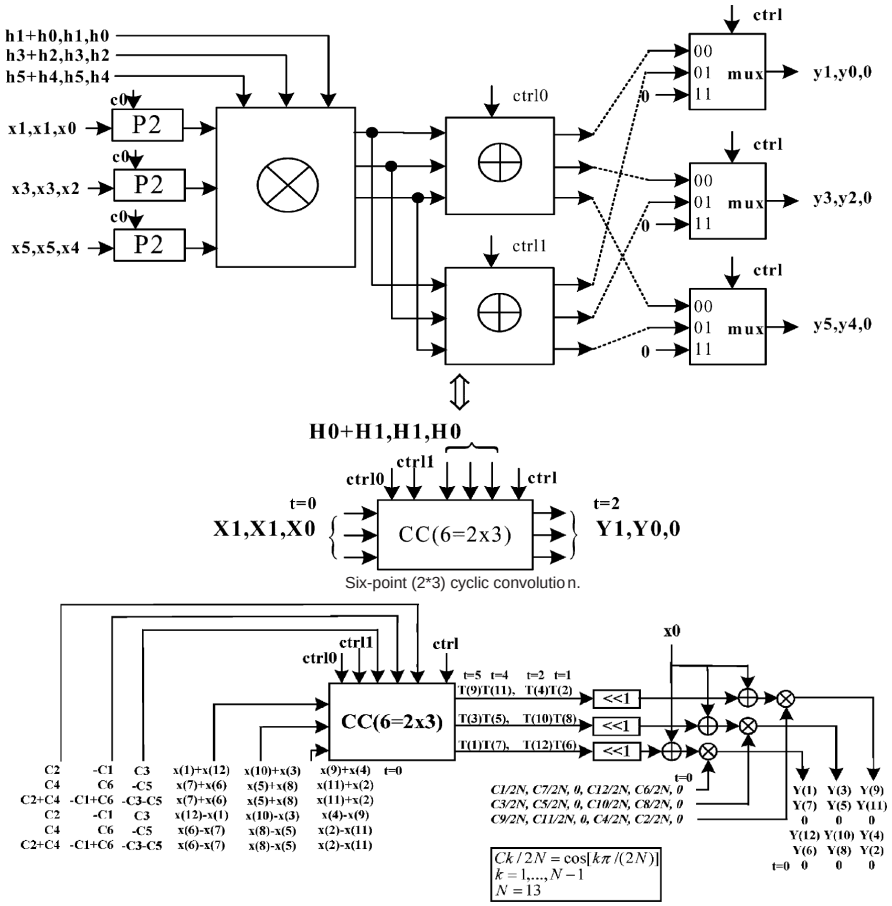


Figure 2.21: Thirteen-point DCT using six point ($6 = 3 \times 2$) fast cyclic convolution [19]

The proposed fast cyclic convolution algorithm is applied to the implementation of prime length DCT , the proposed high-throughput implementation of 1297 length DCT design saves 1216 (94%) multiplications, 282 (22%) additions, and 4792 (74%) delay elements compared with those of recently proposed systolic array based algorithms. Furthermore, the proposed algorithm can run at a speed that is 1.5 times of previous designs, it also requires less I/O cost as long as the word length L is less than 20 bits.

FPGA Implementation of Convolution using Pipelining

In [20], a variable kernel convolution implementation with three different architectures metrologies is presented. The first uses sequential streaming, the second uses pipelining and the third solution is call convolve and gather. The proposed architecture for convolution using pipelining is presented in Fig. 2.22

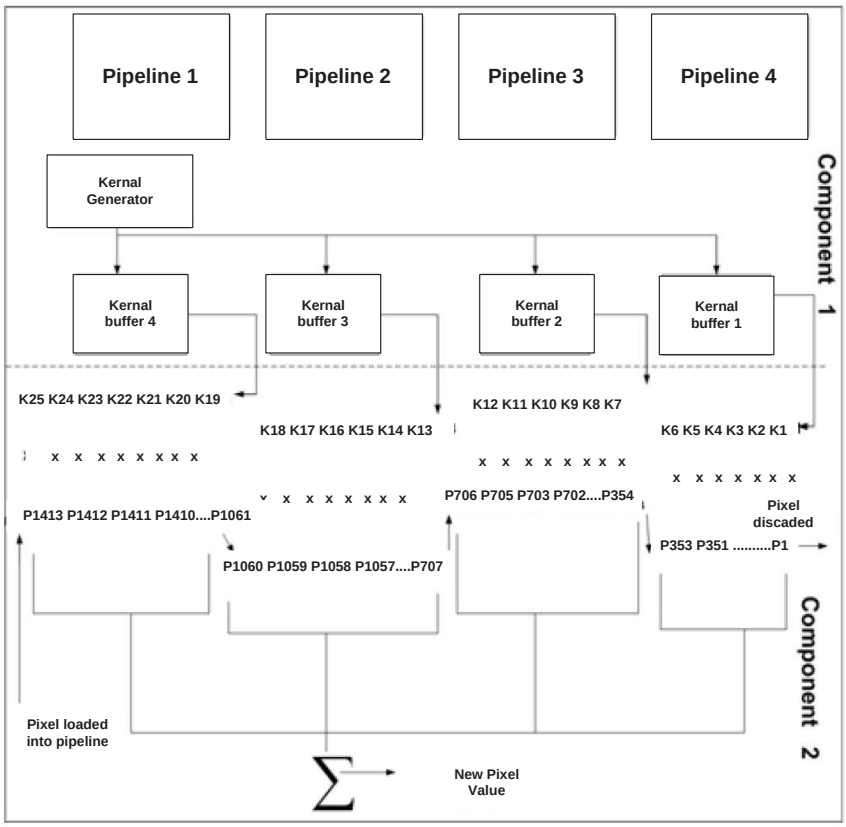


Figure 2.22: Architecture for convolution using pipelining [20]

In this work, the pipeline computation is distributed between four pipeline stages by allocating 354 pixels (P1060 to P1413) and seven kernel coefficients (K19-K25) to the first pipeline, 353 pixels (P707 to P1059) and six kernel coefficients (K13-K18) to the second pipeline, 353 pixels (P354 to P707) and six kernel coefficients (K7-K12) to the third pipeline and 353 pixels (P1 to P353) and six kernel coefficients (K1-K6) to the fourth pipeline. In this setup, kernel coefficients are loaded in from above (i.e. from the kernel buffers), while pixels are loaded in from the left. Each clock cycle a new pixel is loaded in, all the pixels in the pipeline are shifted right and the rightmost pixel is discarded. Every clock cycle the pixels are multiplied with the corresponding kernel coefficients and the

results of multiplications are added together to generate the new convolved pixel. This proposed design was also implemented on Xilinx Virtex 2Pro (XC2VP70-6) chip with maximum frequency of 167 MHz and utilized 3501 slices.

Efficient FPGA Implementation of Convolution

In [21] a direct method of reducing convolution time using hardware computing and implementations of discrete linear convolution of two finite length sequences ($N \times N$) are presented. This implementation method is realized by simplifying the convolution building blocks. The basic building block for proposed design is shown in Fig. 2.23.

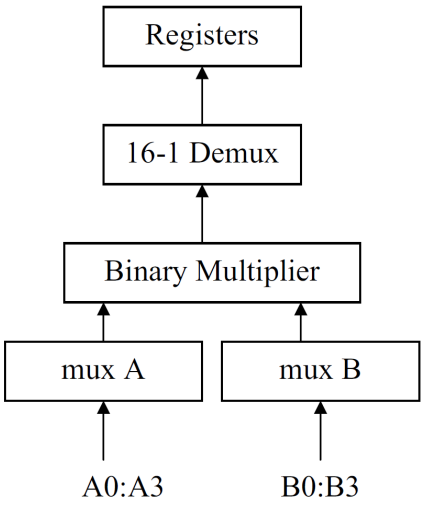


Figure 2.23: Implementation block diagram [21]

In this paper, the circuit deals with two signals having N values. It considers the two numbers like two arrays having four locations each to store values. Each array is fed into a quadruple 4×1 Mux separately. The selection of values is done by selection switches of each Mux. The selected values go into the array multiplier and from there they are routed into Parallel Load Registers (PLR) through a 1X16 Demux. The proposed implementation uses a modified hierarchical design approach, which efficiently and accurately speeds up computation; reduces power, hardware resources, and area significantly. The efficiency of the proposed convolution circuit is tested by embedding it in a top level FPGA. Simulation and comparison with different design approaches show that the circuit saves almost 35% of area and it is four times faster than what is implemented in previous work. In addition, the presented circuit uses less power consumption and has a delay of 20ns from input to

output using 32nm process library.

2.3 Power Modelling

Power dissipation has become a key design issue in FPGA based architectures. In this section, a brief description of various existing FPGA power modelling techniques operating at different design abstraction levels have been presented in the following sections.

2.3.1 High-Level Power Estimation of FPGA

In [22] a power estimation and exploration methodology based on high level power modeling approach of reconfigurable devices such as FPGA is presented. In order to address the different abstraction levels and the various targets, a global methodology is proposed here to elaborate suitable models. In this work the FPGA power estimation can be obtained at early stage of the design process. The proposed model for FPGA design flow is shown in Fig. 2.24.

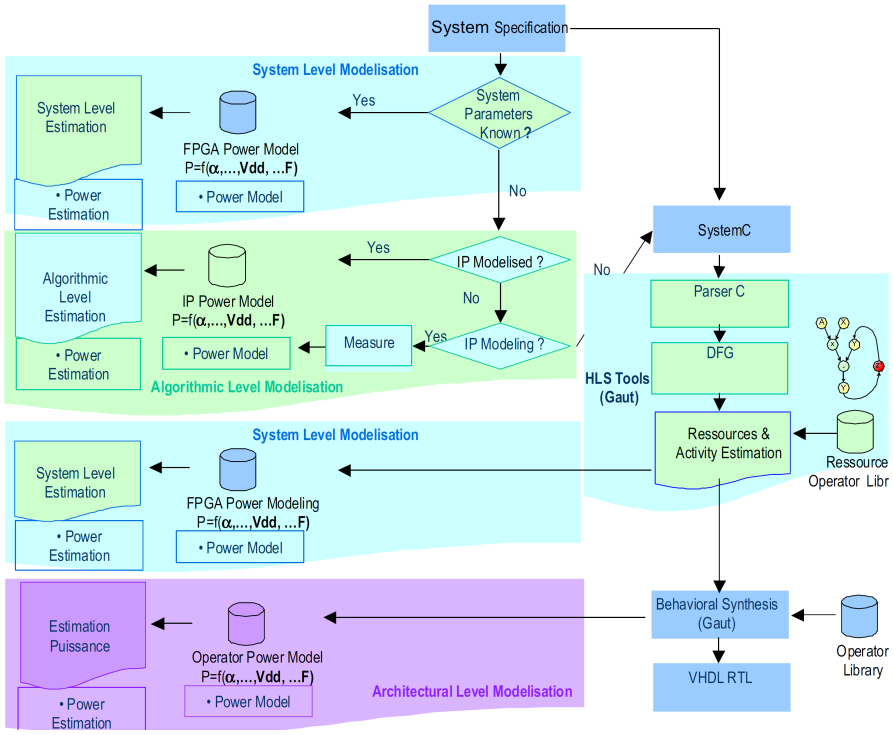


Figure 2.24: Model for FPGA design flow estimation [22]

In this paper different ways of getting an estimation from a given specification are presented in order to refine the specification while exploring the design space and optimizing the

power consumption of the targeted FPGA implementation. According to [22] the designers are able to estimate the power in three different level such as system, algorithm and architecture level. In other words this methodology relies on multi-level power models. The choice of the estimation level depends on the models and information at disposal. In [22] indicates that this approach gives relatively good results. The average error goes from about 3% to 30% depending on the considered level. The maximum error never exceeds 37%. Indeed, the precision appears better in the algorithmic level than the system level.

2.3.2 Functional Level Power Analysis Modeling on Complex Processors

In [23] a high level consumption estimation methodology and its associated tool, SoftExplorer are presented. The estimation methodology uses a functional modeling of the processor combined with a parametric model to allow the designer to estimate the power consumption when the embedded software is executed on the target. The block diagram for the methodology is presented in Fig. 2.25.

The first step, divides the processor architecture into different functional blocks and sub-blocks. Then, the relevant consumption parameters are selected as the significant links between these blocks. There are two types of parameter: algorithmic parameter values depend on the executed algorithm (typically the cache miss rate) and architectural parameter values depend on the processor configuration settled by the designer (typically the clock frequency). The second step is the characterization of the processor power consumption when the parameters vary. Characterization can be performed either by measurements or by simulation. This work shows that the average error of the SoftExplorer results against the physical measurements is about 2.4%.

2.3.3 Power Model for Field-Programmable Gate Arrays

In [24,81] a detailed and flexible power model which has been integrated in the widely used Versatile Place and Route (VPR) CAD tool. This power model estimates the dynamic, short-circuit, and leakage power consumed by FPGAs. It is the first flexible power model developed to evaluate architectural tradeoffs and the efficiency of power-aware CAD tools for a variety of FPGA architectures, and is freely available for noncommercial use. The

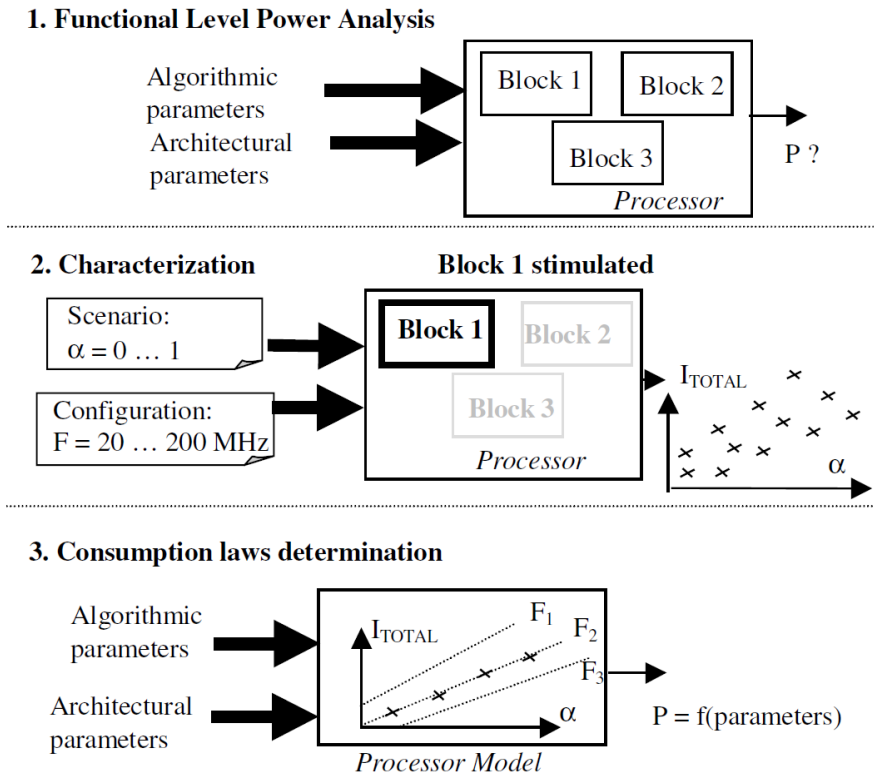


Figure 2.25: Modeling methodology block diagram [23]

model is flexible, in that it can estimate the power for a wide variety of FPGA architectures, and it is fast, in that it does not require extensive simulation, meaning it can be used to explore a large architectural space. In addition, in this work it has been show that how this model can be used to investigate the impact of various architectural parameters on the energy consumed by the FPGA, focusing on the segment length, switch block topology, look up table size, and cluster size. The authors claim the model is flexible, in that

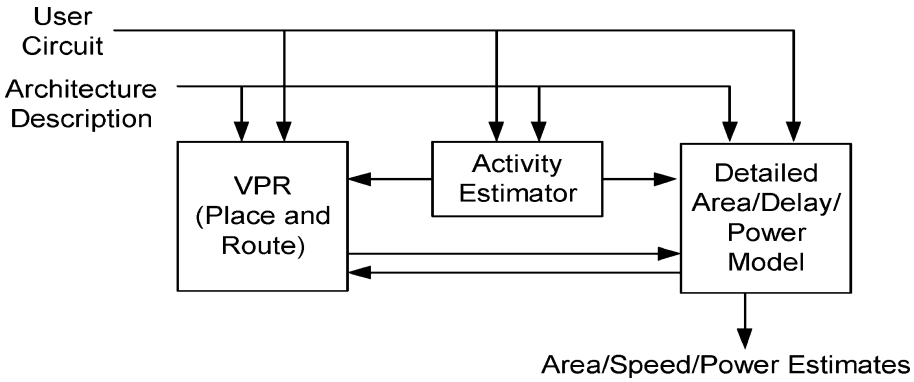


Figure 2.26: Modified VPR framework used for power modelling [24]

it can estimate the power for a wide variety of FPGA architectures, and it is fast, in

that it does not require extensive simulation, meaning it can be used to explore a large architectural space. The model has two modules: an activity generation module and a power estimation module. The first module employs the transition density model to determine the switching activities inside the circuit. The second module estimates the power consumption at the transistor level. The model was calibrated using Simulation Program with Integrated Circuit Emphasis (HSPICE) with the technology parameters from a 1.8V, 0.18-μ CMOS technology. Dynamic power estimation includes routing and logic block components. Short circuit power is assumed to be 10% of total dynamic power. HSPICE simulations for NMOS and PMOS transistors have been carried out to determine effective leakage power. The block diagram of the proposed model is shown in Fig. 2.26.

2.3.4 High-Level Power Modelling of CPLDs and FPGAs

In [25] a high-level power modelling technique to estimate the power consumption of reconfigurable devices such as Complex Programmable Logic Devices (CPLDs) and FPGAs is presented . The development of models is based on input and output signal statistics to estimate the internal power consumption of FPGAs. Input signal modelling is based on the determination of optimal signal partitioning, probability estimation, transition density, signal space correlation followed by output signal transition density. Models for differently-configured circuits are based on the same power macromodel template. To achieve tradeoff between accuracy and efficiency, an adaptive regression method is used to tackle the problem of biased training sequences. In this paper, experimental results indicate that the average relative error is only 3.1% compared to low-level FPGA power simulation methods. The overall process flow diagram is presented in Fig. 2.27.

2.3.5 Macromodels for High Level Area and Power Estimation on FPGAs

In [82] a high-level power model equation based area and power macro-models for various RTL level operators such as adders, multipliers, and logical operators is presented. These models are derived by actual synthesis of these RTL operators using back-end logic synthesis and PAR tools. The presented area estimation technique is based on high-level compile time estimation of the areas of the Control-Data Flow Graph (CDFG) nodes. Each CDFG node represents an operator and is parameterised with the bit-widths of the inputs (such

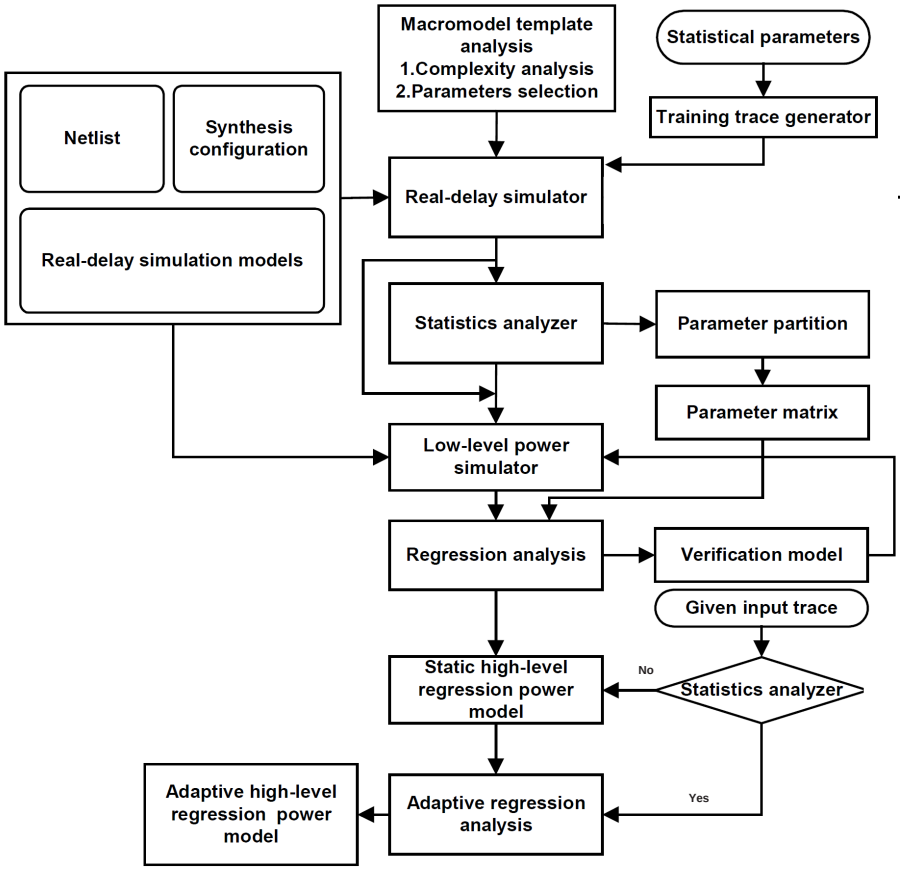


Figure 2.27: High level power macromodelling for reconfigurable hardware [25]

as N-bit adders and multipliers) and characterising the results obtained from post layout to take into account route-through. The proposed power macromodelling approach in [82] is based on average input signal probability P_{in}, the average input transition density D_{in} and the average input spatial correlation S_{in} as the candidates of input metrics. Input bit width N is also taken into account. This platform independent modelling approach has varying levels of accuracy (depends on the core that is modelled) and is not scalable. The block diagram of proposed model is shown in Fig. 2.28.

2.3.6 Functional Level Power Analysis and Modeling on IP Cores

In [12, 26], a behavioral functional level power modelling methodology called Functional Level Power Analysis and Modelling (FLPAM) that provides a good trade off between complexity and accuracy, and enables the designer to achieve incremental improvements in power and energy metrics throughout the design process is presented. The underlying

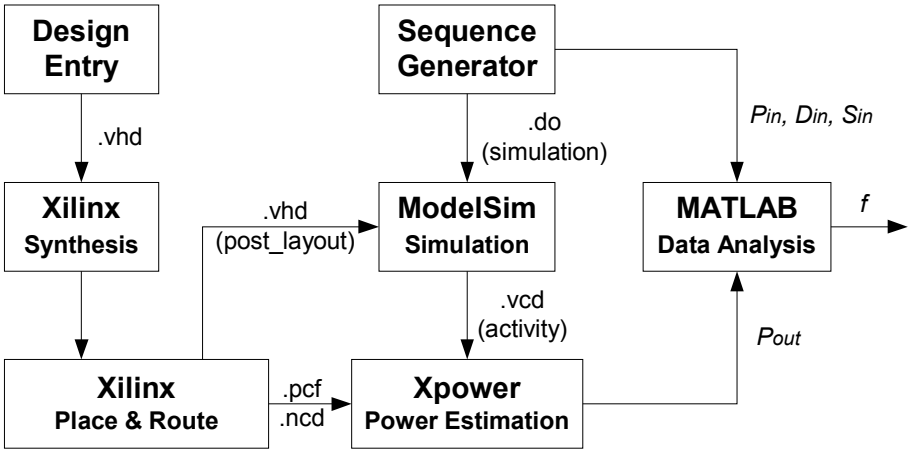


Figure 2.28: Macro-model characterisation procedure

concept of FLPAM is to build a mathematical model that incorporates all the system variables, enabling the user to perform high level estimation of the power and energy metrics of the core for a given set of parameters early on in the design cycle itself. The proposed design flow for FLPAM is presented in Fig. 2.29.

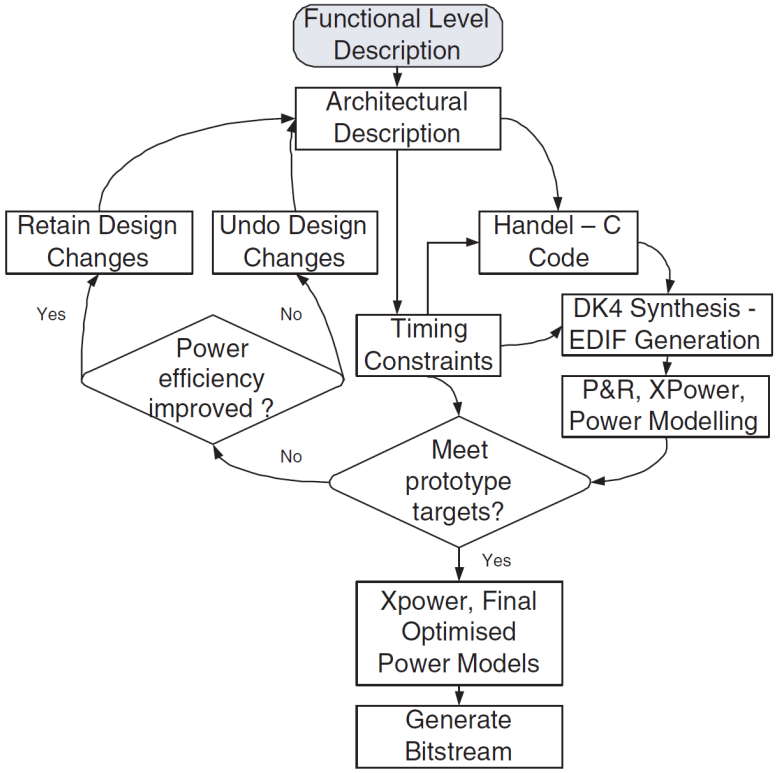


Figure 2.29: Design flow for FLPAM based power and energy optimised design of FPGA cores [26]

The FLPAM methodology has been successfully incorporated into a proposed design flow presented in Fig. 2.29 for obtaining power and energy efficient implementations of FPGA based designs.

2.3.7 Methodology for Dynamic Power Estimation of FPGA Based Designs

In [27] a circuit-level simulations to characterise a simple, coarse-grained FPGA architectural model is presented. The overall proposed power estimation methodology is presented graphically in Fig. 2.30. The dynamic power estimation technique presented in this work involves two processes. First, each resource is characterised to find its effective capacitance. Characterisation is subdivided into global wire modelling and input dependency. Next, power of a given design is estimated by finding the utilisation of each resource and determining its switching activity. Power estimation and accuracy evaluation is performed for a set of 14 designs against silicon measurement. The measurements were taken on an internally developed test board hosting an XC3S1000 FPGA, a mid sized device with 1,920 CLBs. The avarage error was 18% from the measured value. Furthermore, the evaluation has been preformed on Spartan-3 and the results find that in the 90nm FPGA, routing resources and clock trees account for 84% of the total power. Compared to earlier studies, that the clock tree consumes more power and contributes about 39% of the total power.

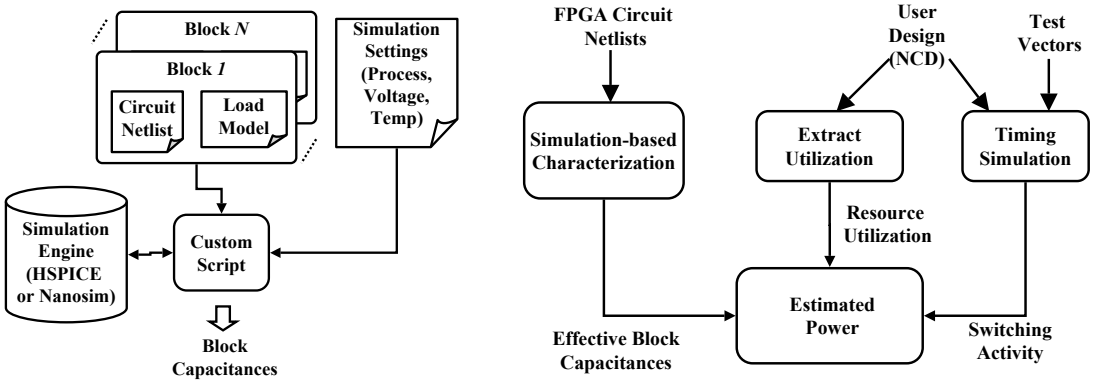


Figure 2.30: High level FPGA power estimation methodology [27]

2.3.8 Post Synthesis Level Power Modelling of FPGAs

In [28] a methodology and tool suite capable of modelling the power consumption of an FPGA design at the post synthesis, or EDIF, level is presented. It is suggested that modelling at this level has the following advantages: Firstly, early power feedback in the design flow. Secondly power results are displayed at a high level, closer to the logical design entry point. Finally, the elimination of bulky, low-level timing accurate simulation and stimulus files. These three aspects allow a designer to quickly and easily generate power estimates, relate the results back to their original logical level design entry and explore design trade-off scenarios. The power modelling approach in this work consists of: 1) developing a tool infrastructure to support synthesis level simulation and circuit queries and 2) developing a synthesis-level power model. This modelling approach has a high accuracy of 97%, but is not scalable and not platform independent. The proposed power podeling tool infrastructure diagram is shown in Fig. 2.31.

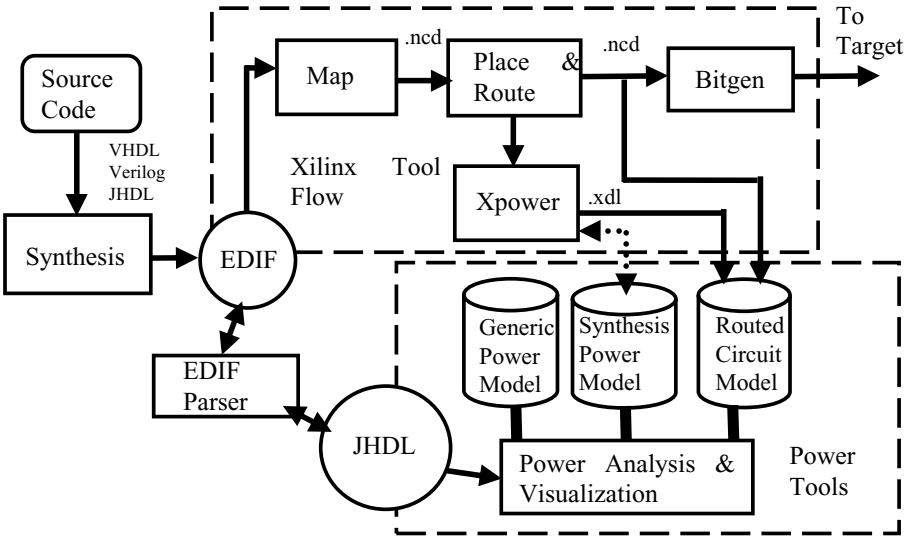


Figure 2.31: Power modeling tool infrastructure [28]

2.3.9 Power Estimation for Cycle-Accurate Functional Descriptions of Hardware

In [29] a methodology for Cycle-Accurate Functional Descriptions (CAFD) power estimation that combines the accuracy achieved by power estimation at the structural RTL with the efficiency of cycle accurate functional simulation by viewing a CAFD as an abstraction of a specific is presented. This work describes that for a given CAFD, corresponding

simulation testbench and a power model library (generated once for each fabrication technology, using well-known characterisation techniques) for RTL components preprocessed is first carried out in order to enable easier back-annotation of RTL information. Virtual component instantiation and idle cycle analysis is then performed resulting in an RTL-aware CAFD which is is co-simulated with the power model library to determine average power. An adaptive state-based sampling technique is used to optimise the allocation of sampling probabilities to different control states for improved representation of states with a higher time-variance of power. It is claimed that the accuracy of this architectural level modelling technique is high. Although it is scalable and parameterisable, the key disadvantage is the unsuitability of this modelling approach for IP core macromodelling. The diagrammatic representation of the modelling methodology is shown in Fig. 2.32.

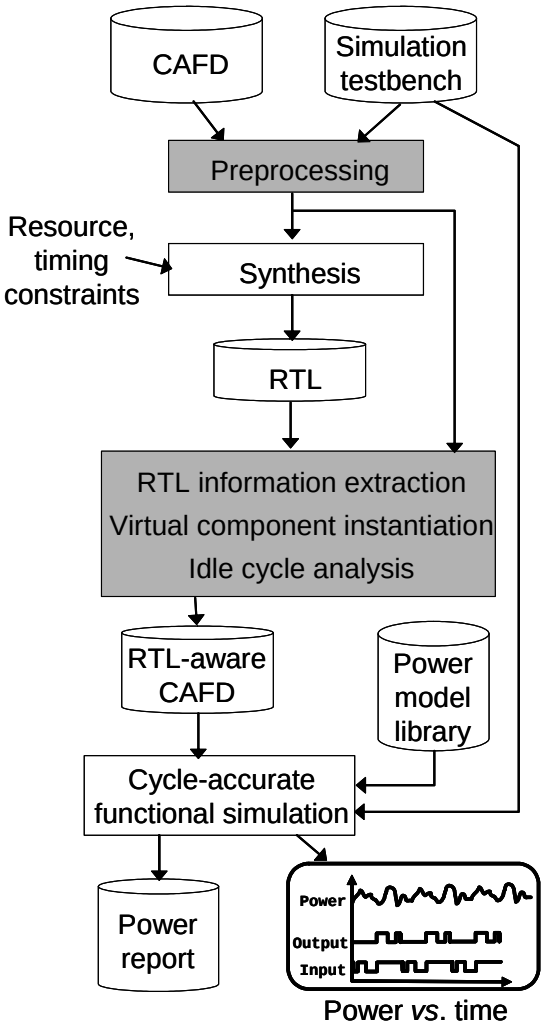


Figure 2.32: Overview of the CAFD power estimation methodology [29]

2.3.10 Power Estimation Technique for FPGAs

In [30] empirical prediction models for these parameters, suitable for use in power aware layout synthesis, early power estimation/planning, and other applications is presented. In this work the authors examine how switching activity on a net changes when delays are zero (zero delay activity) versus when logic delays are considered (logic delay activity) versus when both logic and routing delays are considered (routed delay activity). In this work a novel approach for prelayout activity prediction that estimates a nets routed delay activity using only zero or logic delay activity values, along with structural and functional circuit properties is described. The experimental results in this work show the proposed prediction models work well given the noise limitations. The process flow diagram for proposed design is presented in Fig. 2.33.

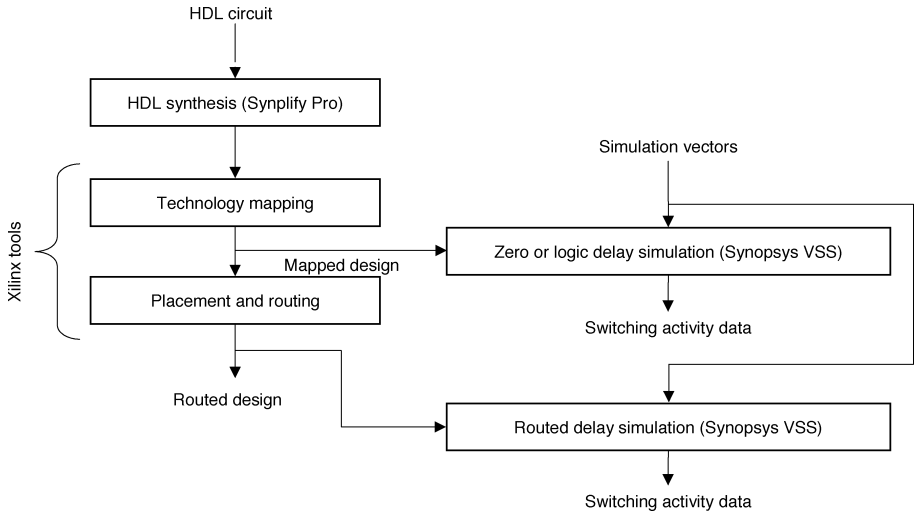


Figure 2.33: CAD flow for activity analysis [30]

In this work it has been shown that for capacitance prediction the prediction accuracy is improved by considering aspects of the FPGA interconnect architecture in addition to generic parameters, such as net fanout and bounding box perimeter length. It is also demonstrated that there is an inherent variability (noise) in the switching activity and capacitance of nets that limits the accuracy attainable in prediction. The proposed prediction model is validated on the Xilinx Virtex-II PRO FPGA family.

2.3.11 Power Modelling and Characteristics of Field Programmable Gate Arrays

In [31] a mixed-level power model that combines switch-level models for interconnects and macromodels for LUTs considering both dynamic and leakage power is developed. Gate-level netlists back-annotated with postlayout capacitances and delays are generated and cycle accurate power simulation is performed using the mixed-level power model. The resulting power analysis framework is named as FPGA EVA-LP2. This work experiments

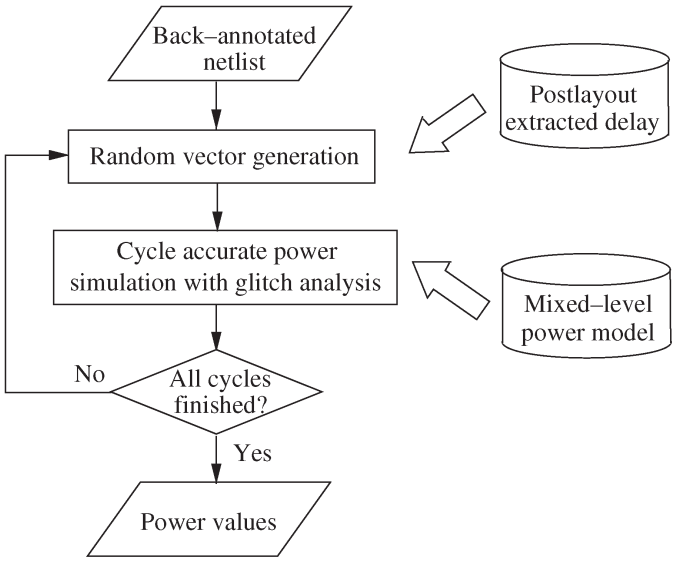


Figure 2.34: Overall power calculation [31]

show that FPGA EVA-LP2 achieves high fidelity compared to SPICE simulation, and the absolute error is merely 8% on average. This work shows that FPGA EVA-LP2 can be used to examine the power impact of FPGA circuits, architectures, and CAD algorithms. It is shown that interconnect power is dominant and leakage power is significant in nanometer technologies. In addition, tuning cluster and LUT sizes lead to 1.7 times energy difference and 0.8 times delay difference between FPGA architectures, and FPGA area and power are reduced at the same time by tuning the cluster and LUT sizes. The overall process flow diagram for proposed model ia shown Fig. 2.34. This mixed level modelling approach has a medium accuracy level of 92% and it is scalable. However, it is not platform independent.

2.4 Limitation of Existing Work and Research Opportunities

As it can be seen from the preceding sections, there has been extensive research on the hardware implementation of multiresolution algorithms, there still remains plenty of scope for further research in exploiting reconfigurable computing for the various applications specially medical images and algorithms that have been addressed. Medical image processing is one of the hottest topic in the field of image processing applications and there is a high demand for an efficient implementation. Take these issues into consideration, our group which has been led by Dr Amira are concentrating on software/hardware implementation of medical image application by using MAAs and other appropriate algorithms. The major limitations of the existing work can be identified as follows:

- Multiresolution algorithms are used in many image processing applications but limited work has been carried out on FPGA implementation and optimisation of medical imaging based on multiresolution algorithms;
- The design optimisation of multiresolution algorithms has not been considered as a holistic challenge balancing the demands for power, performance, area etc;
- In recent work the most of the effort towards the design and hardware implementation (in the form of VLSI and FPGA) of wavelet transforms has been concentrated on the orthonormal wavelet family. One of the main reasons for this is that orthonormal wavelets were the first functions to be implemented in the form of filter banks. Even though, HWT is very easy and simple to implement;
- Although impressive image processing performance has been achieved with FRIT, the complexity of their implementation still remains as a heavy burden on standard microprocessors where large amounts of data have to be processed. Surveying the literature, not many FPGA-based implementation has been found for Ridgelet transform. Therefore, the design of high-performance architectures and their FPGA implementation for Ridgelet transform is strategic for applications that require real-time performances;
- Surveying the literature also shows that some VLSI implementations of DST and

DCT based on cyclic convolution have been carried out, but few FPGA based implementations for cyclic convolution have been found;

- Literature review shows that power awareness has not been a major consideration in existing implementations of FPGA based design. Although, there is a very high demand for power efficient design;
- Literature survey also shows that many of power modelling tools are available for ASICs (VLSI, in general), and only limited solutions are available for FPGAs; and
- Existing power modelling tools for FPGAs may have some features such as high accuracy, low/mixed/high level, platform independence, scalability in varying measures; but generally not all features at the same time.

Based on the limitations of existing work, the main contribution in this work presented in this thesis can be summarised as follows:

- To develop novel architectures for MAAs using advanced arithmetic techniques and design methodologies through the optimisation strategies at various abstraction levels;

  - To design and implement scalable, parameterisable, efficient FPGA based 1-D and 2D multiresolution IP cores suitable for use in both general and medical imaging applications;

  - To explore and implement efficiently HWT on FPGAs using factorisation methodologies;

  - To implement a novel FRIT architecture efficiently on FPGAs using pipelining and parallelism techniques;

- To develop and implement cyclic convolution on FPGAs using parallelism and systolisation which can be integrated in multiresolution architectures as the main building block for efficient computation;
- To investigate and apply optimisation strategies at various abstraction levels (system, algorithmic and architectural level) and to analyse the effectiveness of these techniques for performance enhancement and power reduction;
- To investigate the best performance trade-offs such as area/speed for the FPGA implementations of these cores; and

- To evaluate futher the accuracy of a high level power modelling for the proposed custom IP cores such as HWT FRIT and etc.

2.5 Conclusions

This chapter reviews a number of significant architectures and systems suitable for MMAs implemented on different FPGA platforms using different design methodologies. In addition, the advantages and drawbacks of the existing architectures have also been highlighted. Existing power modelling techniques at various levels for FPGA based designs have also been reviewed. It is the aim of the research work presented in this thesis to address the limitations presented in the previous section through efficient implementations, power aware design and evaluation of power modelling methodology. Efficient architectures for HWT based on DA and factorisation methodologies will be presented in the next chapter.

Chapter 3

Efficient Implementation of HWT using Sparse Matrix Factorisation

3.1 Introduction

In the past two decades, there has been an ever increasing amount of interest in wavelet transform. Wavelet transform is an emerging signal processing technique that can be used to represent real life non-stationary signals with high efficiency. Indeed, the wavelet transform is gaining the momentum to become an alternative contrivances to traditional time frequency representation techniques. Efficient hardware implementation and acceleration of the algorithms used in these applications are becoming very important due to the amount of time required for its processing and the real need for embedded solutions in most of the advanced image and vision systems. Images can be transformed into large matrices which require a large amount of memory, high power dissipation and computationally intensive. For example a typical 640 x 480 color image has nearly a million elements to store, manipulating or storing intensive images require huge amount of time, which leads to an efficient hardware acceleration.

The development of advanced and fast reconfigurable hardware in form of Field Programable Gate Arrays (FPGAs) brings a huge interest in real time image processing. FPGAs can perform mathematical operation on an entire vector or matrix at the same time. There are different arithmetic techniques and design metrologies such as Distributed Arithmetic (DA) and systolic design used to implement image processing algorithms efficiently on FPGA. Even though, these techniques help the designers to implement efficiently the

large images on FPGA, but the issue about power consumption still remains a big concern due to advancement of FPGA. Researchers are working round the clock to address this issue in a efficient way.

In this chapter two novel factorisation methodologies for the implementation of the HWT and their impact on FPGA implementation using DA principles are presented. The hardware embedded solution for HWT will be used to accelerate the process in the proposed system. Fig. 3.1 illustrates the overall process of the proposed segmentation system, where medical image volume is acquired from a Positron Emission Tomography (PET) scanner. The proposed HWT methodologies are used to segment the acquired image (slice by slice). The process of segmentation is implemented on FPGA for acceleration using the proposed architectures. The detailed infromation about the proposed architectures is explores in the following sections.

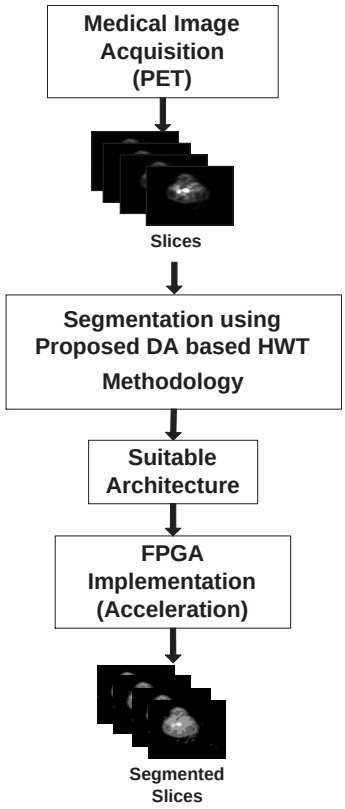


Figure 3.1: Proposed system for medical image segmentation using HWT with FPGA acceleration

The rest of this chapter is organised as follows. A review of HWT and DA are presented in Section 3.2 and 3.3. The proposed HWT Factorisation Methodologies (HWTFM1 and HWTFM2) are present in Section 3.4. Two novel architectures for proposed HWTFM1

and HWTFM2 based on DA principles are presented in Section 3.5. The efficient FPGA implementation results and analysis for proposed algorithms are presented in Section 3.6. Concluding remarks are presented in Section 3.7.

3.2 Haar Wavelet Transform: A Review

The first recorded mention of what is now called a "wavelet" seems to be in 1909, in a thesis by Alfred Haar. An image is represented as a Two Dimensional (2D) array of coefficients, each coefficient representing the brightness level in that point. In wavelet analysis, a signal can be separated into approximations or averages and detail or coefficients. Averages are the high-scale, low frequency components of the signal. The details are the low scale, high frequency components. From all wavelet filters Haar wavelet is one of the simplest possible wavelet transform. The disadvantage of the Haar wavelet is that it is not continues and therefore not differentiable.

3.2.1 HWT Decomposition Methods

Image is presented mathematically in a form of matrix. Haar wavelet uses a method for manipulating these matrices called averaging and differencing in other word it is called decomposition. There are two types of decomposition methodologies standard and non-standard.

The Standard Decomposition

- First apply the one-dimensional wavelet transform to each row of pixel values.
- Treat these transformed rows as if they were themselves an image and apply the one-dimensional transform to each column

The Nonstandard Decomposition

- Apply one step of horizontal pair wise averaging and differencing on the pixel values in each row of the image.
- Apply vertical pair wise averaging and differencing to each column of the result. The Nonstandard Decomposition

Averaging and differencing is very effective method but the calculations can quickly become quite tedious for large matrix sizes. For instance, if we apply averaging and differencing on a vector of 256×1, it needs 8 level decompositions, but if we apply the same process it on 256×256, it requires 2048 level of decompositions. It shows the difficulty of averaging and differencing for large matrix. The process od averaging and differencing are described below:

- For the first step, take the average of each pair of pixels from the first row (original image) and places the results in the first four position of our new row. The remaining four numbers are the differences of the first element in each pair and its corresponding average. These numbers are called detail coefficients. The result of the first step therefore contains four average and four detail coefficients.
- The same procedure is applied to the first four components of the new row resulting in two new average and their corresponding detail coefficients. The remaining four are carried directly down from previous step.
- The same method is applied to the remaining two pairs and the last six are directly carried down from previous step. The new string after three steps ends up with one average and seven detail coefficients. For vector of 8 elements 3 steps are required so the calculation for larger matrix it would very difficult and time consuming. By applying linear algebra a general [83] formula can be generated to average and difference (decompose) any matrix.

3.3 Mathematical Background

HWT is a matrix-vector based operation and can be formulated as follows [84]:

$$I = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \tag{3.1}$$

$$H = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3.2}$$

$$Q = I \times H \tag{3.3}$$

$$Q = ((IH)^T H)^T = H^T IH \tag{3.4}$$

$$I = (H^{-1})^T Q H^{-1} \tag{3.5}$$

where I is a 2x2 input matrix, H contains the Haar coefficients and Q is the transformed matrix. Eq. 3.4 and Eq. 3.5 show the transposed and reconstructed matrices respectively.

3.4 Distribute Arithmetic: A Review

DA provides an efficient method of computing vector or matrix multiplication by means of bit level rearrangement of the multiply accumulate process. DA distributes arithmetic operations rather than grouping them as multipliers do. Conventional DA, called ROM-based DA, decomposes the variable input of the inner product to bit level in order to generate pre-computed data. ROM partitioning techniques for the efficient implementation of DA for large vector sizes have been presented in [85, 86]. The basic operations required for performing DA-based inner product are a sequence of ROM look ups, addition, subtraction and shift operations of the input data sequence. All of these functions are efficiently mapped to FPGA structures. DA exploits parallelism (at the vector level) and pipelining (at the bit level) and is highly suitable for FPGA implementation due to their fine grained device fabric, massive parallelism capabilities, register rich architecture that enables efficient implementation of ROM structures in LUTs [2, 87]. The superior performance and hardware efficient nature of DA when compared to conventional arithmetic has been suitably demonstrated in the implementation of various algorithms such as algorithms based on matrix-vector multiplication which is presented in [8, 76].

Since HWT is a matrix-vector based operation, DA is a suitable arithmetic technics for it to implements the HWT on FPGAs due to its efficiency of mechanization, however it becomes slow particularly when deadline with large transformations because of its bit serial nature. Although by using some modification like partitioning and bit pairing its performance can be significantly increased.

Let the input and transformed data be represented by two vector X and Y of size K, respectively. Then Y can be written as follows:

$$Y = \sum_{k=1}^{K} A_k X_k(n) \tag{3.6}$$

where A is the constant coefficients, X is the input data word, and Y is the transformed vector (output data). If X_k is considered to be in the form of a scaled 2's complement binary number, it can be represented as:

$$X_k = -b_{k0} + \sum_{n=1}^{W-1} b_{kn} 2^{-n} \tag{3.7}$$

where b_{kn} is the n^{th} bit of X_k "which can be '0', or '1'",$-b_{k0}$ is the sign bit, $b_{k,W-1}$ is the Least Significant Bit (LSB), and W is the word length. Substituting Eq.3.7 into 3.6, we get the following:

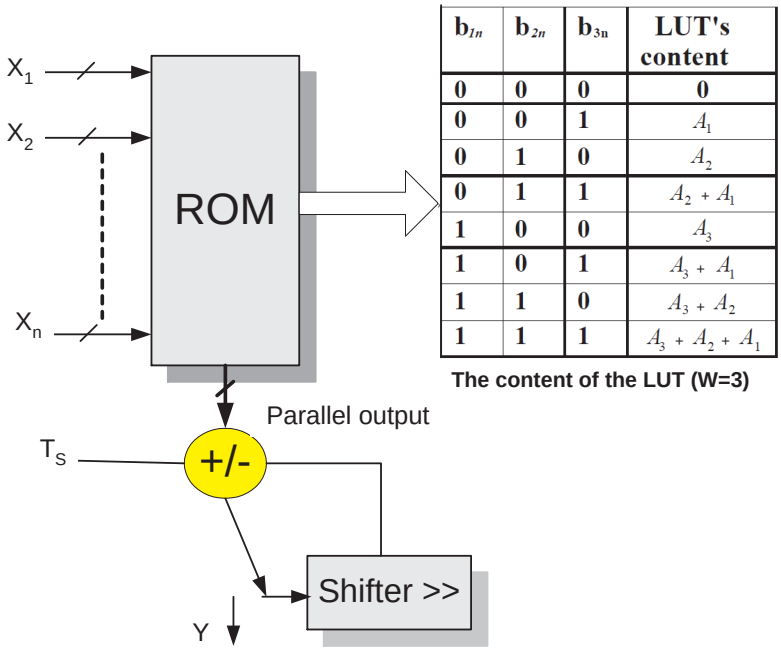
$$Y = \sum_{k=1}^{K} A_k \left[-b_{k0} + \sum_{n=1}^{W-1} b_{kn} 2^{-n} \right] \tag{3.8}$$

By rearranging the order of summations in order to convert the conventional sum of products into a "distributed" form, we get:

$$Y = \sum_{n=1}^{W-1} \left[\sum_{k=1}^{K} A_k b_{kn} \right] 2^{-n} + \sum_{k=1}^{K} A_k (-b_{k0}) \tag{3.9}$$

The term $\sum_{k=1}^{K} A_k b_{kn}$ can have only 2^k possible values, which makes it possible to pre-compute and store these values in a ROM. By addressing this ROM through N cycles using the input data and performing simple shift-accumulate operations. The final mathematical DA equation can be represented into hardware architecture which is shown in Fig. 3.2

Fig. 3.2 shows a typical hardware architecture which consists of ROM (made of LUT), adder-structure and a shift-accumulator. Using bit-serial input data with the Least Significant Bit (LSB) one bit of each inputs are used to address the LUT. After each shift operation one output bit is generated in a serial form. This is repeated until the sign bit is high, then a subtraction is performed. The most significant part is generated in a parallel form. When the size of the inner products increases the ROM area increases exponentially and becomes impracticably large, even when using ROM partitions.



b_{1n}	b_{2n}	b_{3n}	LUT's content
0	0	0	0
0	0	1	A_1
0	1	0	A_2
0	1	1	$A_2 + A_1$
1	0	0	A_3
1	0	1	$A_3 + A_1$
1	1	0	$A_3 + A_2$
1	1	1	$A_3 + A_2 + A_1$

The content of the LUT (W=3)

Figure 3.2: DA Harware Architecture

3.5 HWT Factorisation Methodologies

HWT is based on averaging and differencing neighbor pixels in which the computation time can be increased when dealing with large matrices. There would be a high demand for a fast and efficient implementation. However, applying direct HWT coefficients generated by linear algebra can decrease the computational time, but it will not be efficient in terms of hardware implementation due to so many arithmetic calculations. Since we are dealing with matrices, two factorisation methodologies can be proposed to factorise the HWT coefficients shown in Eq.3.2 to increase the sparsity of the matrix (increase number of zeros in the matrix to reduce the use of logics in term of hardware). The factorisation methodologies are called HWT Factorisation Method 1 (HWTFM1) and HWT Factorisation Method 2 (HWTFM2) which are described in the following sections.

3.5.1 The Proposed HWT Factorisation Method 1

The process of averaging and differencing systematically averages two neighboring pixels in a given matrix, then finds the difference between the same pixels. The averaging and differencing method is very effective, but the calculations can quickly become quite tedious for larger images. By introducing the Haar coefficients through linear algebra

in the form of the matrix shown in Eq.3.2, we could easily simplify the transformation process. However, it is a suitable method for software implementation but it is still not ideal for hardware implementation due to so many arithmetic calculations. To generate the Haar transformation process suitable for hardware implementation, Eq.3.2 is factorised into two matrices to introduce more zeros in the matrices to reduce the use of logic in the hardware process. The factorisations have been formulated in generalized term which can be used for any matrix size. It is worth mentioning that the matrix row and column have to be based-2 to decompose using HWT decomposition methodologies. For instance, if the matrix row is 8×1 then it is based-2 format (2^3) and it can be decomposed with 3 level of HWT decompositions. The generalized factorisation methodology for HWTFM1 is shown in Fig. 3.3.

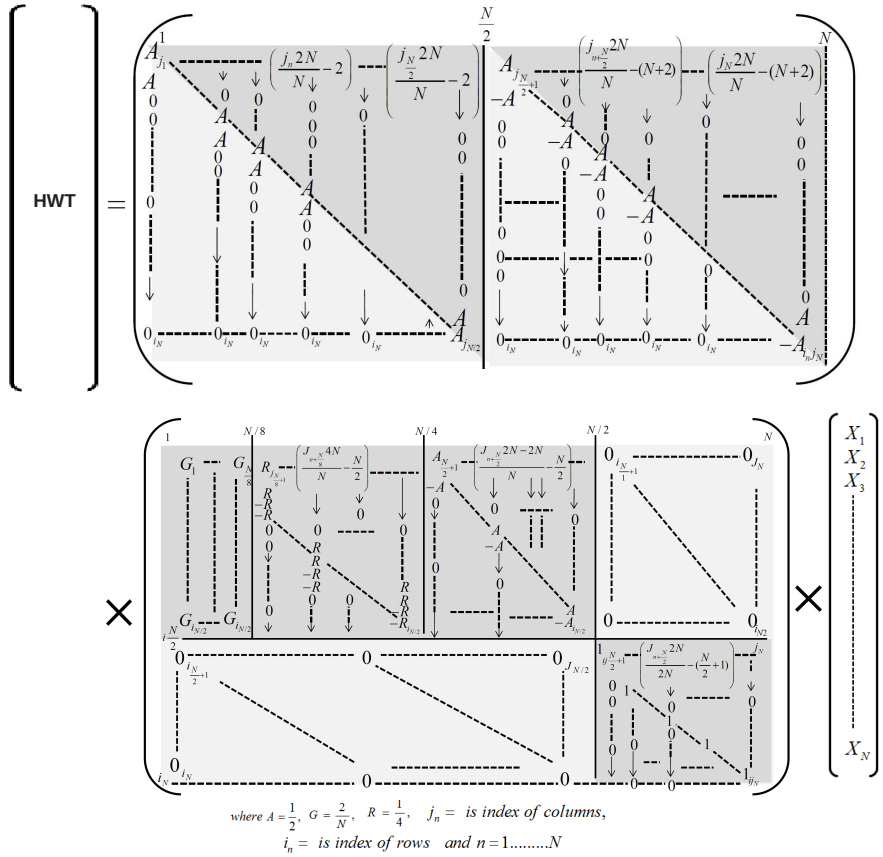


Figure 3.3: Generalised formulation for HWT factorisation method 1

It can be seen from, Eq.3.2 that it is divided into two matrices as shown in Fig. 3.3, where A is constant number (0.5) and N is the size of matrix. Each formula in the matrix represents the number of zeros in the j^{th} column of $N \times N$ matrix. It is worth mentioning that that formulas represent number of zeros until the first non zero element appears in the

same column. From Fig. 3.3 it is clear that the upper matrix divided into two halves and all non zero elements appear in a diagonal format in each half. It is not the case in lower half, the matrix divided into two halves and the first half divided in three halves where each half has different number of zeros and different non zero elements. This generalised HWTFM1 has been tested with different N size with positive outcomes.

3.5.2 The proposed HWT Factorisation Method 2

The same procedure as HWTFM1 is applied to the HWTFM2 to increases number of zeros, the only difference is that the second matrix has more zeros than the second matrix of MWTFM1. The advantages and disadvantages of these two methods will be explained in the next section where the architectures for these methods are proposed. The generalized HWTFM2 is shown in Fig.3.4. As it has been presented for HWTFM1, A is a constant

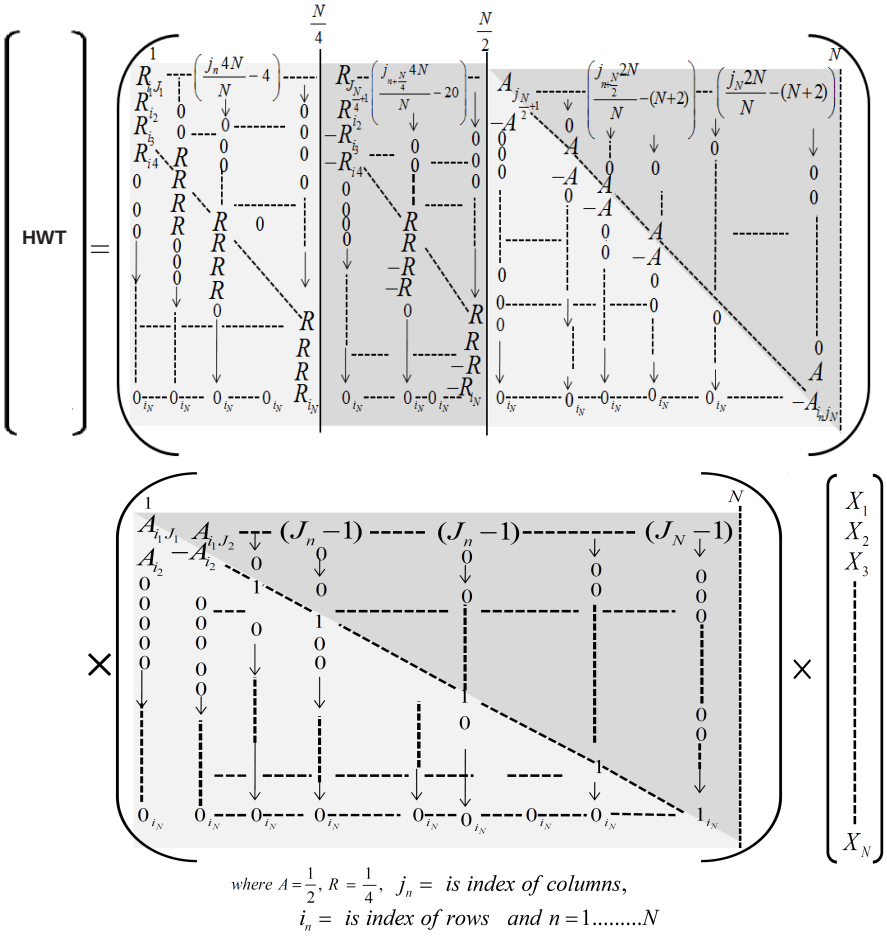


Figure 3.4: Generalised formulation for HWT factorisation method 2

number and N is the size of the matrix. All formulas in both matrices represent the

number of zeros in the Jth column of each matrix. HWTFM2 has also been tested with different N size with positive outcome. It is clear from the HWTFM2 that second matrix has more zeros which help to reduce the arithmetic calculation even more in the case of hardware logic. For clarification of HWTFM1 and HWTFM2 a simple matrix where N=8 is used as an example which is shown in the following equations.

The HWTFM1 when N=8

$$\begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \\ Y_8 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} \end{pmatrix} \times$$

$$\times \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{4} & -\frac{1}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & -\frac{1}{4} & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \end{pmatrix} \tag{3.10}$$

$$\begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \\ Y_8 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} \end{pmatrix} \times \begin{pmatrix} \frac{1}{4}(X_1 + X_2 + 2X_3) \\ \frac{1}{4}(X_1 + X_2 - 2X_3) \\ \frac{1}{4}(X_1 - X_2 + 2X_4) \\ \frac{1}{4}(X_1 - X_2 - 2X_4) \\ X_5 \\ X_6 \\ X_7 \\ X_8 \end{pmatrix} \tag{3.11}$$

The HWTFM2 when N=8

$$\begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \\ Y_8 \end{pmatrix} = \begin{pmatrix} \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{4} & 0 & -\frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & 0 & -\frac{1}{4} & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & -\frac{1}{2} & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & 0 & -\frac{1}{2} \end{pmatrix} \times$$

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \end{pmatrix} \tag{3.12}$$

$$\begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \\ Y_8 \end{pmatrix} = \begin{pmatrix} \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{4} & 0 & -\frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & 0 & -\frac{1}{4} & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & -\frac{1}{2} & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & 0 & -\frac{1}{2} \end{pmatrix} \times \begin{pmatrix} \frac{1}{2}(X_1 + X_2) \\ \frac{1}{2}(X_1 - X_2) \\ (X_3) \\ (X_4) \\ X_5 \\ X_6 \\ X_7 \\ X_8 \end{pmatrix} \tag{3.13}$$

3.6 Proposed Architectures for HWTFM1 and HWTFM2

In this section, two novel architectures have been proposed for the implementation of HWTFM1 and HWTFM2 which are followed by a tabular comparison of the various parameters of the proposed architectures with other existing architectures in place.

3.6.1 Proposed Architectures for HWTFM1

The proposed architecture for the HWTFM1 (Arch1) is illustrated in Fig. 3.5. The inputs to the circuit are fed in bit-serial fashion from the input port. The arithmetic calculation will be processed by a number of Processor Elements (PEs) and ROMs. In

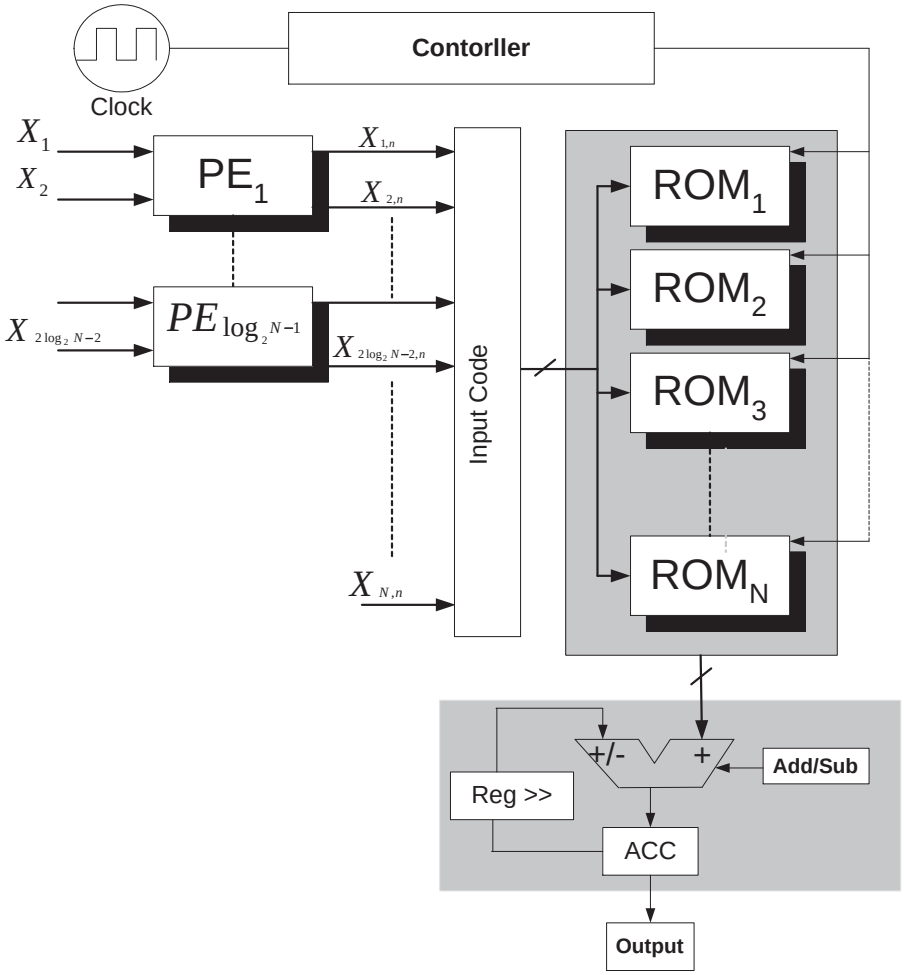


Figure 3.5: Proposed architecture for HWTFM1

the case of Arch1, $log_2N - 1$ PEs are used as address generator, where each PE contains

ADD/SUB and Shift Register which are used to generate the elements of second matrix in Eq.3.11. Another N ROMs are used to store the values of first matrix in Eq.3.11. Since the architecture is based on DA principles, it is worth mentioning that each ROM stores the content of each row in Eq.3.11. The values in row_1 will be stored in ROM_1 and the

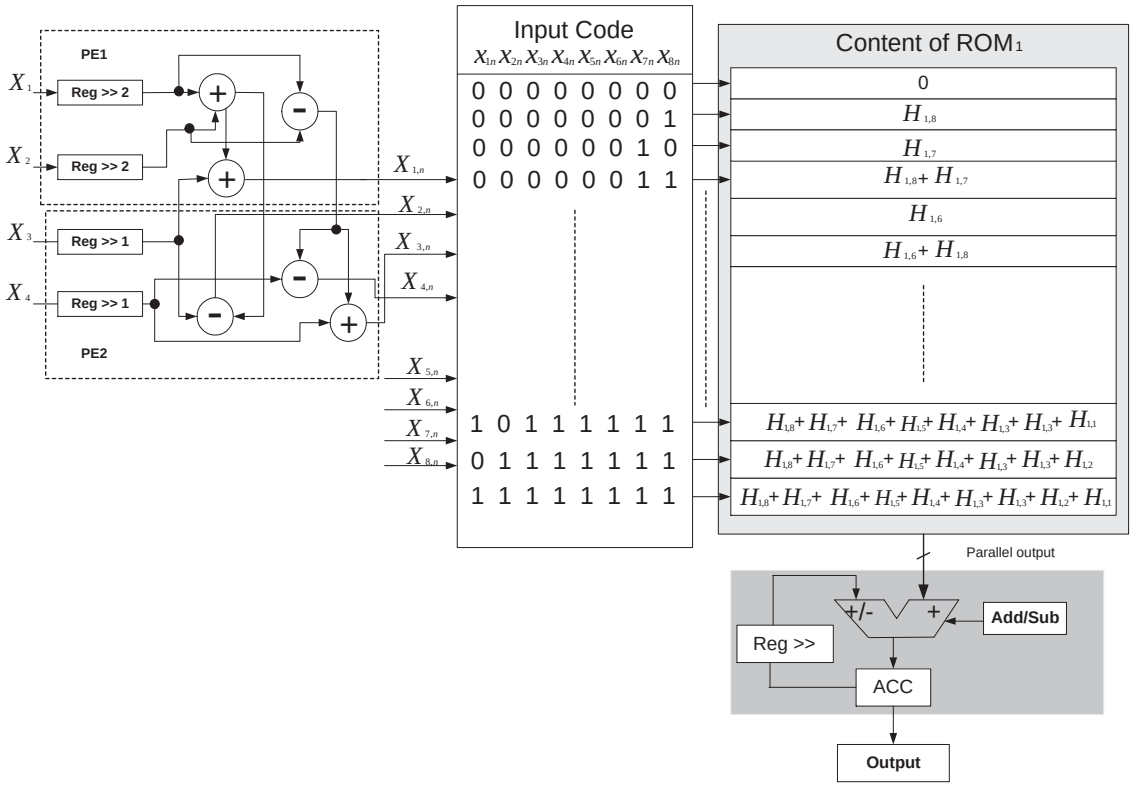


Figure 3.6: HWTFM1 structure for a vector when N is 8 and W is 8

value of row_N will be stored in ROM_N. The controller is used to select the ROMs and based on the inputs (column) the decoder selects the corresponding ROM addresses and its contents. For instances, if inputs are from column_2 the content of ROM_2 will be selected then the DA principles will be applied to calculate the outputs. A delay of two clock cycles has been incorporated in the architecture to synchronise inputs $(X_1.....X_{2log_2-2})$ with inputs$((X_{2log_2-2)+1}.....X_N)$ then another N clock cycles are needed to complete the overall operation. For clarification purposes we explore the address generator and the content of ROM for HWTFM1 vector when$N$ is 8 and W is 8. The detailed process is illustrated in fig. 3.6. It can be seen from fig. 3.6 that the ROM_1 contains the values of first row in the Haar coefficients, a 8 bit decoder is used to access the appropriate contents of ROM_1. The same procedure is applied to other Haar coefficients.

3.6.2 Proposed Architectures for HWTFM2

The proposed architecture for the HWTFM2 (Arch2) is illustrated in Fig. 3.7. The design for Arch2 is similar to Arch1, with some advantages and drawbacks. In the case of Arch1 $log_2N - 1$ PEs are required to generate the elements of the second matrix as shown in Eq.3.11, but it is not the case for Arch2, where there will be no PE to generate the elements of the input matrix, only two shift registers and one ADD/SUB are required to generate the first two elements of the input matrix. It means Arch2 uses less logic for address generator than Arch1. On the other hand the number of non zeros elements

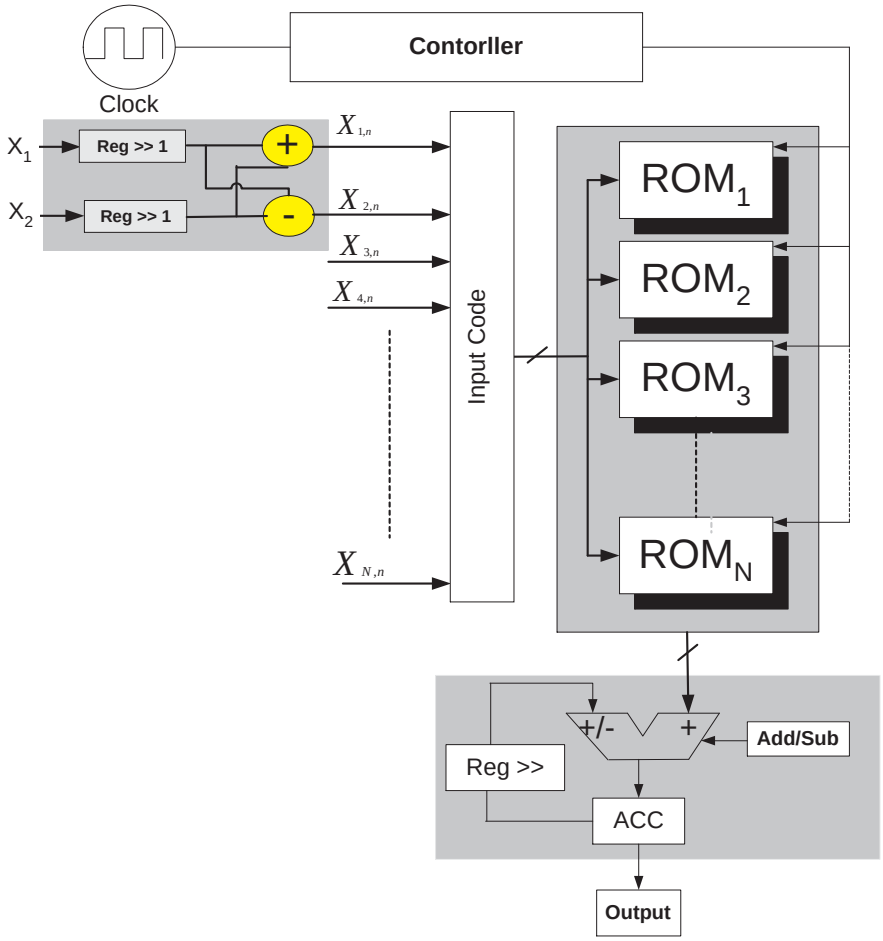


Figure 3.7: Proposed architecture for HWTFM2

increases in the first matrix of Eq.3.13 which are the constant content of the ROMs in Arch2, in other word there will be more non zero elements to be stored in the ROM of Arch2 compare to Arch1 and more arithmetic calculations are required to process the outputs. The computation time will be the same as the Arch1.

3.6.3 Comparison with Existing Architectures

A direct implementation of DA based HWT without any modification requires (in the case of $W = 8$ and $N = 8$) 24 additions and subtractions where N and W are the transform size and word length respectively. Applying HWTFM1 reduces the additions and subtractions to 16 and in the case of HWTFM2 it reduces to 18. Design parameters such as Time Complexity (TC) and Area Complexity (AC) of the proposed design and other existing architectures are presented in Table 4.5. It is worth noting that since Arch1 and Arch2 are DA based on DA, the TA depends on W. It is worth mention that the number of bits used to represent each word in the victor and not victor length. Thus, the latency of the design and number of clock cycle remains constant for a given input wordlength for all victor size. In this case, when $W = 8$, Arch2 requires ten clock cycles *(W+2)* to produce (Y_1). During the first two clock cycles (X_1) and (X_2) are fed in the additional circuitry to synchronies with other inputs $(X_3....X_N)$ and during the last eight clock cycles output (Y_1) is produced. The AC of both architectures are based on the ROMs (LUTs, ADD/SUB and shift registers) and the additional circuitry which synchronies the inputs. It is worth noting that the AC depends on victor (N) and it increases exponentially with (N). It is clear from Table 4.5 that the proposed architectures outperform the other existing architectures in term of AC, TC and ADDs/SUBs.

Table 3.1: Comparison of the design parameters with other existing architectures (where N is transform size and W is the word length)

	TC	**AC**	**Add/Sub**
Proposed 1	$O(W+2)$	$O((log_2N-1)+N)$	$3N-8$
Proposed 2	$O(W+2)$	$O(N+1)$	$(Nlog_2N-6)$
[12] DA Based	$O(2(W+1))$	$O(2^{N/2})$	$Nlog_2N$
[88] SA B. Level	$(2N-1)(W+log_2N)$	$O(N^2)$	NA
[11] Serial DA	$O(2(W+1))$	$O(2^{N/2})$	NA
[11] Parallel DA	$O((N+1)W)$	$O(2N)$	NA
[89] Bit L. Vector	$O(2N)$	$O(N^2)$	NA

The comparison of logic elements for different methodologies are shown in Table 3.2. It is clear that the proposed methodologies increases number of zeros in the content of ROM and reduces the logic resource used such as adders and subtractors. It is worth mentioning that by applying the proposed methodologies increased the number of zeros in the content of ROM by 33% and 20% respectively compare to direct implementation of HWT without any modification. Proposed mythologies also reduced the arithmetics (add/sub) by 33%

and 25%. In other word the proposed methodologies increase zeros and use less logic resource in terms of hardware than other existing methodologies.

Table 3.2: Comparison of logic elements used in the proposed methodologies when $N = 8$ and $W = 8$

Method	N	W	Add/Sub	Zeros in LUTs
Applying HWT	8	8	24	32
Proposed HWTFM1	8	8	16	48
Proposed HWTFM2	8	8	18	40
[12]	8	8	24	32

3.6.4 HWT Host Application for FPGA

The communication between FPGA and user is always been complex, to resolve this issue a host application has been generated for user to provide an easy access to FPGA. The Host application provides a user interface with pull down menus to interactively select the type of implementation and level of decomposition for chosen image. Based on the requirements of the system architect the image can be selected, then FPGA configuration bitstream is loaded on the FPGA initiating the transformation process. All the current RC boards (including RC10) come with some type of Flash memory on board, flash memory is used to store data which can be accessed by FPGA and user (Host) through USB controller. The flash can be accessed through read and write functions and the data stores in the array of indexes of 1 to 254 (1 byte per index). The Host application sends the data to flash memory. On completion of input data, the Host sends a signal to the FPGA and releases control over the flash memory. The FPGA takes control over the flash memory, and reads the image from the flash. Each image block is transformed to the HWT stored back into flash. After the entire image is processed, control over the flash memory is released by the FPGA back to the Host. The transformed image is read by the Host from flash memory and displayed in the user interface. A snapshot of the host-FPGA system is shown in Fig. 3.8. It is worth mentioning that the image size is used to store in the flash was 256×256 and the maximum size flash memory on the RC10 board is 16 MB.

3.7 FPGA Implementation Results and Analysis

The proposed HWTFM1 and HWTFM2 are implemented on FPGA using Handel C [35] to gain maximum benefit in performance from the target hardware by using its parallel

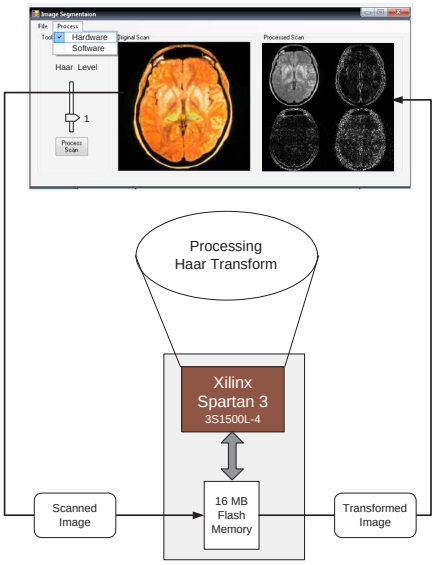


Figure 3.8: Host application for FPGA implementation of HWT

constructs. The detailed explanations of Handel-C performance is provided in Appendix-B. The FPGA implementation process using Handel-c is shown in Fig. 3.9. The data are stored in sperate locations of the SRAMs in parallel by using Handel C parallelism instruction. Once the data are written in the SRAMs, FPGA reads data from the SRAMs in parallel fashion and execute them, after the execution, the data are restored in different locations of SRAMs which is ready to be used by the user.

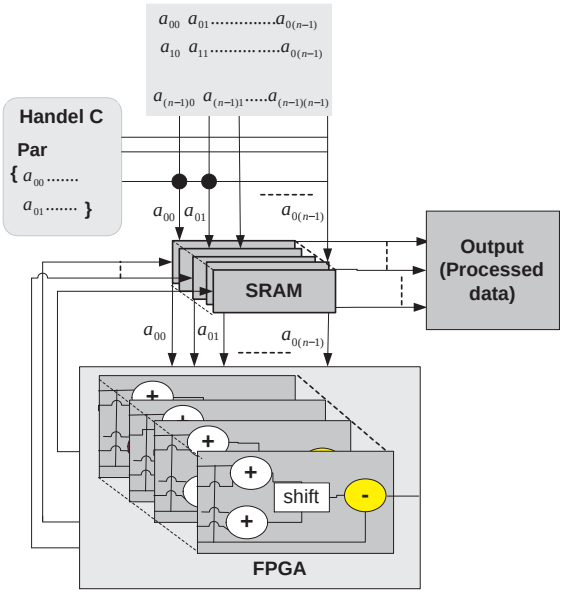


Figure 3.9: Architecture for FPGA implementation of HWTFM2 using Handel C parallelism

In order to evaluate the performance of the proposed HWT hardware implementations, the

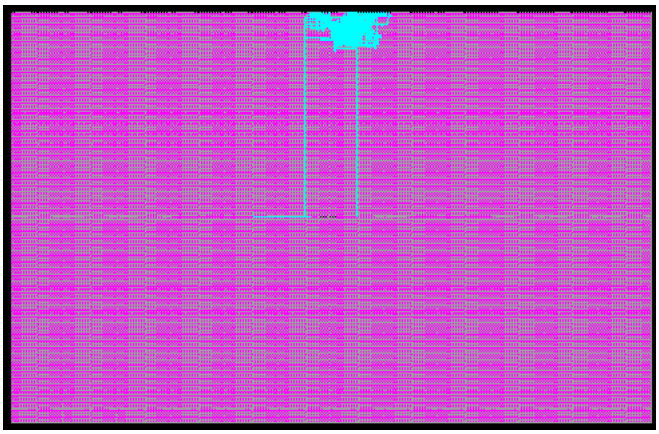
designs have been implemented on the latest FGPA Virtex-5 (XC5VLX330) [90] devices and low power FPGA Spartan3 (3s1500L) [91]. In addition, the results have been compared with other existing implementation results in place. The implementation results for proposed 2D HWTFM2 with N=16 is shown in Table 3.3. It is clear from Table 3.3 that the implementation results for the proposed architecture outperformed the other existing implementations results in terms of area and maximum frequency. The chip layout for proposed HWTFM2 IP core on different FPGA is shown in Fig. 3.10. It is worth mentioning that the the computation times for implementation 16 x 16 image was 185.791 ms using MATLAB and 6.49 ns using FPGA. The question may occur that the comparison between MATLAB and FPGA is not fair, but it is not the case, we simply highlight the importance of FPGA in term of image processing acceleration.

Table 3.3: Comparison of implementation results of HWTFM2 with different platforms and existing work

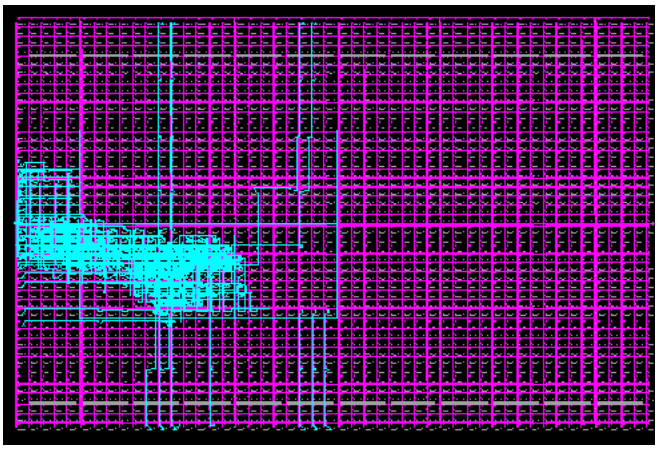
Platforms	**N**	**Area (slices)**	**LUTs**	**I/Os**	**Freq. (MHz)**
Proposed (HWTFM2) Virtex-5	16	186	332	49	154
Proposed (HWTFM2) Virtex-4	16	286	314	49	125
Virtex-4 [92]	16	358	N/A	21	121
Virtex-E [92]	16	335	N/A	21	67
Proposed (HWTFM2) Spartan-3L	16	288	315	48	80

Chip Level Details

After the implementation, the place and route of critical nets and manual pin assignment for the designs have been performed using Xilinx Pinout Area Constraints Editor (PACE) and Floorplanner [2]. This process yields compact and optimised design with short nets and serves two important purposes. Firstly, short nets have lesser propagation delay and up to 25% gains in maximum frequency have been achieved. Second, short nets have lesser parasitic capacitance and DC load and therefore dissipate lesser power than long nets. Manual pin assignment also enables us to locate the I/O pads close to the design area, further aiding the above two criteria. The chip diagram for $N = 16$ is shown in Fig. 3.10.



(a)



(b)

Figure 3.10: Chip layout of Virtex-5 and Spartant-3L for 2D HWTFM2 (a) Chip layout of Virtex-5 for HWTFM2 when the transform size (N) is 16 (b) Chip layout of Spartan-3L for HWTFM2 when the transform size N is 16

3.7.1 Hardware/Software Implementation of HWT on Medical Imaging

HWT has been used in many vital applications including medical image segmentation, due to its multiresolution capability for signal representation. Medical image segmentation is an essential stage in medical image processing. This stage includes significant analysis work by delineating the anatomical structures and discriminating them from image background [93], [94]. More detailed information about HWT segmentation can be found in [95]. The HWT FPGA implementation by using Handel-C is used to implement and segment medical images to highlight the importance of hardware in terms of accelerations. On the other hand, it is also implemented using softwares Central Processing Unit (CPU). The

hardware implementation is coded with Handel C and prototyped on RC10 board which is equipped with Spartan-3 (3S1500L-4) [32]. It is worth mentioning that the implementation of HWT is based on normal standard HWT pseudocode which is presented in [96]. The design flow for both hardware and software implementation is shown in Fig. 3.11.

FPGA Implementation

The scanned medical image is loaded to the Flash memory through Host application, due to the limitation on the RC10 board peripherals the image has been partitioned into a block of 64x64; therefore, the image size of 256x256 is divided into 4 blocks of 64x64 then each block is sent to the FPGA board to be processed and sent back into flash by FPPGA to be read by Host. The design implementation (256x256) occupied 609 FFs, 8,027 LUTs, 4,594 slices and with maximum frequency of 32.519MHz. The hardware segmented medical image applying fist level decomposition is shown in Fig. 3.12 (b,c).

MATLAB Implementation

MATLAB is a high-level technical computing language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran [97]. This interactive environment can be used for algorithm development, data visualization, data analysis, and numeric computation. MATLAB has been used in a wide range of applications, including signal and image processing, communications, control design, test and measurement, financial modeling and analysis, and computational biology. Add-on toolboxes (collections of special-purpose MATLAB functions, available separately) extend the MATLAB environment to solve particular classes of problems in these application areas. Since MATLAB is a suitable environment for image processing, HWT has been also implement using MATLAB for quality purposes with FPGAs implementation. It is worth mentioning that the MATLAB implementation of HWT base on the design flow shown in Fig 3.11. A standard HWT decomposition function (based on averaging and deferencing) which is available in Matlab tool box is used to segment the real PET image. The MATLAB segmented medical image applying first level decomposition is shown in Fig. 3.12 (f,g).

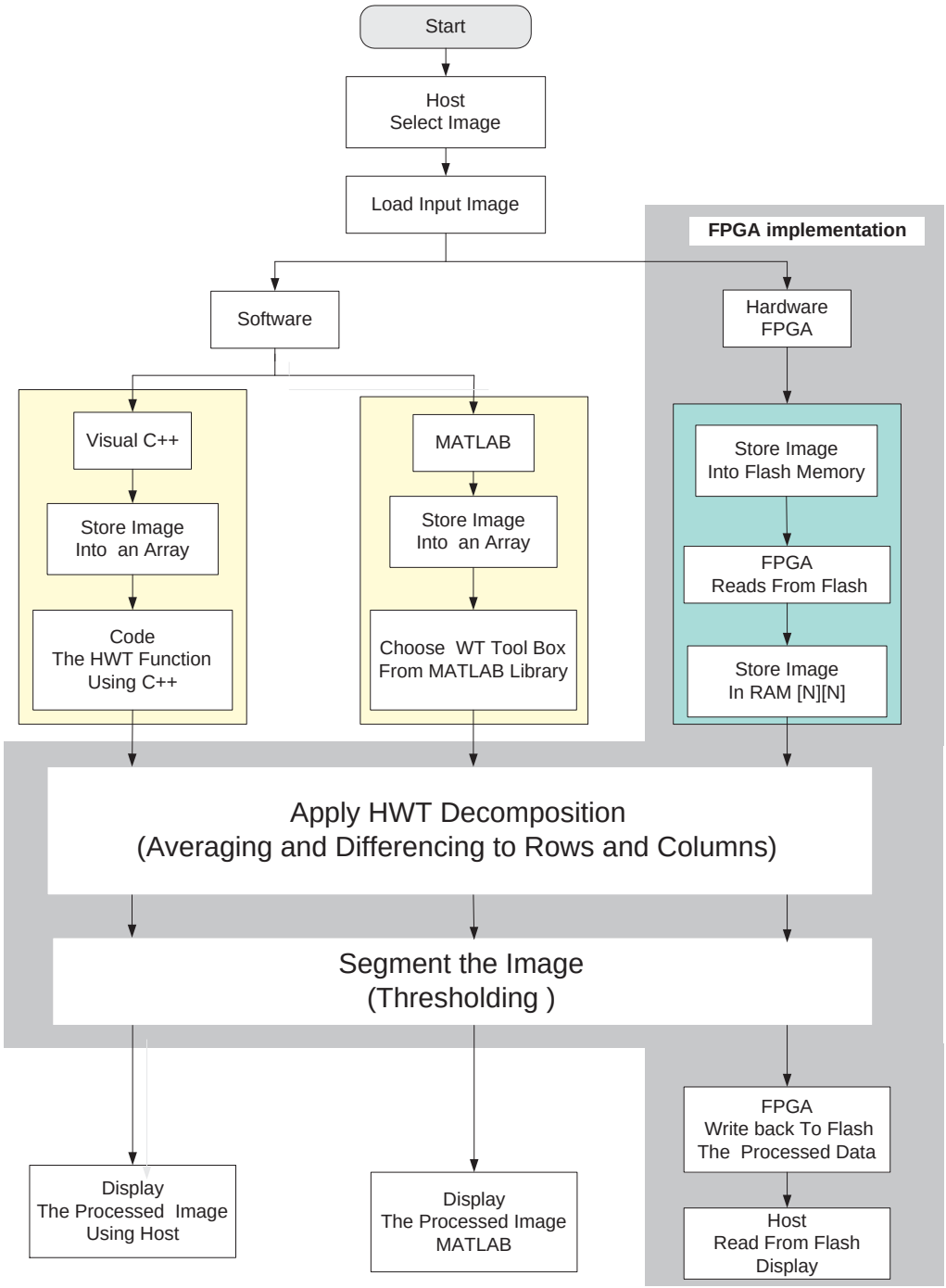


Figure 3.11: The design flow for HWT hardware and software implementation

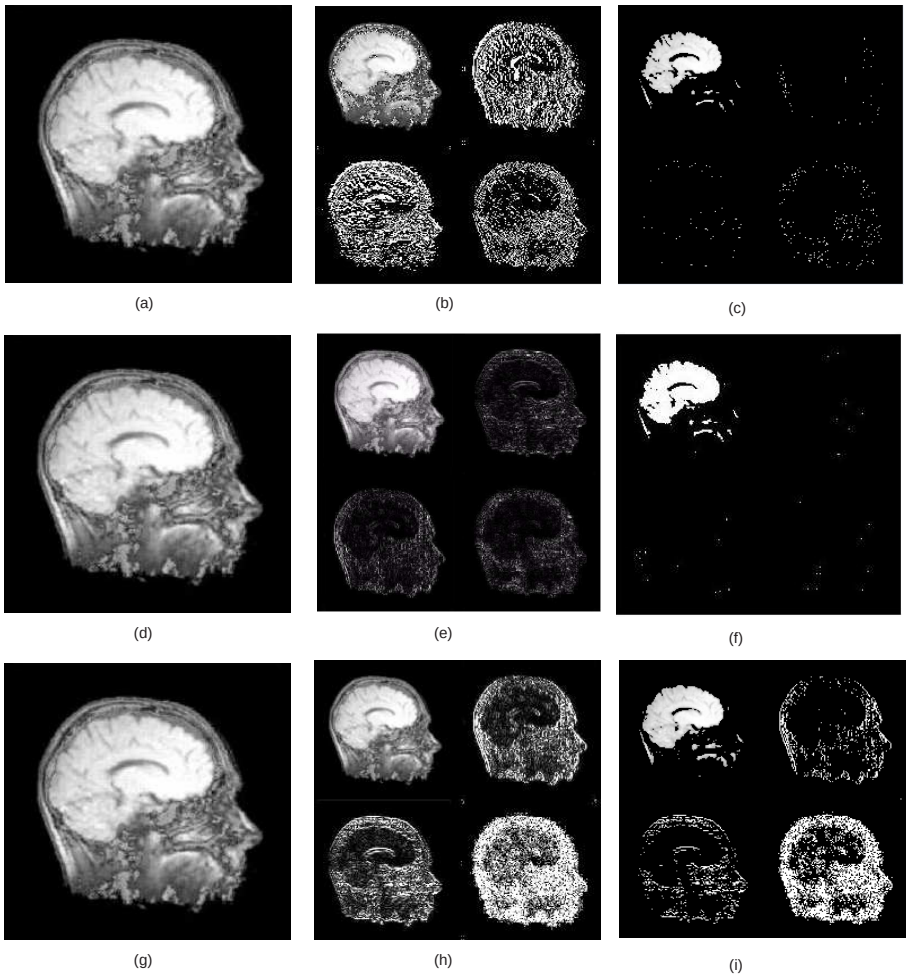


Figure 3.12: (a) Original real PET image (256 x 256) (b) HWT first level of decomposition (c) Segmented image (thresholding) using FPGA (d) Original real PET image (256 x 256) (e) HWT first level of decomposition (f) Segmented image (thresholding) using MATLAB (g) Original real PET image (256 x 256) (h) HWT first level of decomposition (i) Segmented image (thresholding) using Visual C++

Visual C++ Implementations

Visual C++ provides a powerful and flexible development environment for creating Microsoft Windows based and Microsoft .NET based applications. Visual C++ consists of the following components. The Visual C++ Libraries include the industry-standard Active Template Library (ATL), the Microsoft Foundation Class (MFC) libraries, and standard libraries such as the Standard C++ Library, consisting of the iostreams library and the Standard Template Library (STL), and the C Runtime Library (CRT). In addition to conventional graphical user-interface applications, Visual C++ enables developers to build Web applications, smart-client Windows-based applications, and solutions for thin-client and smart-client mobile devices. Throughout our research we have used Visual

C++ as user interface application to send data to FPGAs and read data from FPGAs. Since it is a powerful and flexible development environment and used as user interface application we have decided to carry on the the HWT image segmentation using Viual C++ for quality purposes with MATLAB and FPGA implementation. The Visual C++ coding is based on the HWT pseudocode which is presented in [96]. The Visual C++ segmented medical image applying fist level decomposition is shown in Fig. 3.12 (h,i). Fig. 3.12 shows the first level of decomposition of HWT and segmented image using thresholding on medical image with size of 256 x 256. The PSNR analysis is shown in Table 3.4, it is clear form the Table 3.4 that PSNR values for all implantation are very good. It means the quality of image using hardware/software is the same, but computational time for the first level of decomposition is 212.81 ms using MATLAB and 2630.12 ms using Visual C++ which show a real need for acceleration. In this case hardware implementation

Table 3.4: The PSNR analysis of HWT using FPGA, MATLAB and VC++

Tools	**Image** Size	**HWT 1 level Decom.** PSNR (dB)	**HWT Thresh.** PSNR (dB)
FPGAs	256×256	33.187	32.302
MATLAB	256×256	34.132	33.210
VC++	256×256	33.297	32.823

can be the best solution for the acceleration. The same image is implemented using FPGAs with computation time of 32.30 ns. From these results it is clear that MATLAB computation time is by far faster than Visual C++, but considerably slower than FPGA's computation time. Again it is not fair to compare the computation time between hardware and software, but it is not the case, we simply highlight the importance of hardware (FPGAs) over software in term of image processing acceleration. It is worth mentioning that we have used the same threshold value (190) for all three implementations, it clear from the segmented images and their PSNR valuse that all three outputs are almost the same in terms of quality. In other word, the FPGAs can accelerate the process without losing important information.

3.8 Conclusions

In this Chapter, two HWT factorisation methodologies HWTFM1 and HWTFM2 have been proposed to increase number of zeros and reduce hardware resources. The pro-

posed factorisation methodologies have been formulated in generalized term which can be used for any matrix size. The generalized proposed factorisation methodologies have test on different matrix size with positive outcomes. In addition two efficient and optimised architectures for proposed methodologies based on DA principles have been proposed. The proposed novel architectures are based on DA principles and sparse matrix factorisation technique, and they are suitable for FPGA implementation. The first proposed architectures (HWTFM1) has a time complicity of $O(W+2)$, area complexity of $O((log_2N-1)+N)$ and uses $3N-8$ adders/subtractors. The second proposed architecture HWTFM2 has a time complexity of $O(W+2)$, area complexity of $O(N+1)$ and uses $(Nlog_2N-6)$ adders/subtractors. The evaluation of the architectural results have shown that these architectures outperforms existing architectural results in place. It is worth mentioning that by applying the proposed methodologies the arithmetics calculation (additions/subtractions) is reduced by 33% and 25% respectively compared to direct implementation HWT.

The proposed 2D HWTFM2 when N=16 is implemented on FPGA using Handel-C to gain maximum benefit in performance from the target hardware by using its parallel constructs. In order to evaluate the performance of 2D HWTFM2 hardware implementation, the proposed design has been implemented on the advanced FPGA (Virtex-5) and low power FPGA (Spartan3). The implementation results for the proposed architecture outperformed the other existing implementations results in terms of area and maximum frequency. It is worth mentioning that the the computation times for implementation N=16 image was 185.791 ms using MATLAB and 6.49 ns using FPGA. It highlights the importance of FPGA in term of image processing acceleration. In the next chapter, an architectural level optimisation and efficient FPGA implementation of FRIT will be discussed.

Chapter 4

Architecture Level Optimisation and Efficient FPGA Implementation of Finite Ridgelet Transform

4.1 Introduction

Recently, the ridgelet and curvelet transforms [46–48] have been generating a lot of interest due to their superior performance over wavelets. The wavelet transform has been extensively used in image and video processing during the last ten years. However, it has long been known that the wavelet transform has many limitations when it comes to representing straight lines and edges in image processing. Wavelets are thus more appropriate for the reconstruction of sharp point-like singularities than lines or edges. These shortcomings of wavelets are well addressed by the ridgelet and curvelet transforms, as they extend the functionality of wavelets to higher dimensional singularities, and are effective tools to perform sparse directional analysis. Although impressive image processing performance has been achieved with ridgelet transform, the complexity of its implementation still remains as a heavy burden on standard microprocessors where large amounts of data have to be processed. Therefore, the complexity of ridgelet can be resolved by introducing parallelism and pipelining into hardware implementation. Parallelism can be used to assign different tasks to different concurrent objects in the design, or to speed

up certain iterative operations by performing the same operation on different data-sets concurrently [98, 99]. Parallelism can be exploited in the implementation of matrix operations based transforms such as the Finite Radon Transform (FRAT) where redundant subexpressions can be effectively parallelised. Although additional hardware resources are utilised, but savings can be made in other areas such as minimising buffers and memory elements, reducing frequency to maintain throughput etc.

Pipelining is a well known technique used in Application Specific Integrated Circuits (ASICs) for reducing logic depth and improving throughput at the cost of additional latency. Pipelining is most effective for complex repetitive tasks where each task can be broken down into independent sub-tasks (or stages) which can be executed in a sequential manner. The key advantage of pipelining is the reduction of circuit glitching [100], which is particularly significant in the case of FFPGAs, because of limited availability of programmable interconnects [101]. In this chapter, efficient and optimised architectures for FRAT and Finite Ridgelet Transform (FRIT) by applying principles of parallelism, pipelining and systolisation as appropriate are explored. The proposed architectures are then implemented on different FPGA platforms and the results have been evaluated with other existing work in place.

The rest of this chapter is organised as follows. A brief review for FRAT is presented in Sections 4.2. The proposed architecture for the FRAT and its FPGA implementation are presented in Sections 4.3 and 4.4 respectively. A brief review for FRIT is presented in Section 4.5. The proposed architecture for FRIT and its FPGA implementation are presented in Sections 4.6 and 4.7 respectively. Concluding remarks are presented in Section 4.8

4.2 The Finite Radon Transform: A Brief Review

FRAT was first introduced in [102] as the finite analogue of integration in the continuous Radon Transform (RT), with origins in the field of combinatorics. The mathematical representation of an injective form of the FRAT to ensure invertibility when applied on finite Euclidian planes has been presented in [79]. In recent years the RT has received much attention, due to its ability to transform two dimensional images with lines into a domain of possible line parameters, where each line in the image will give a peak positioned at the corresponding line parameters. This have led to many applications within

image processing and computer vision. The FRAT is defined as summations of image pixels over a certain set of lines. Those lines are defined in a finite geometry in a similar way as the lines for the continuous Radon transform in the Euclidean geometry. RT is an integral transform used to represent an image as a collection of projections along various directions. Sparse representation of image data, especially in images that contain a number of line discontinuities can be effectively achieved using RT. It has enjoyed a position of fundamental importance to many applied problems in mathematics, physical and functional analysis. Applications of RT include seismology, radio astronomy, electron micrography and most famously in tomography [103]. While easy to implement in digital form by discretising the input image, the absence of a corresponding inverse was a key issue. It is worth mentioning that the FRAT is not a discrete version of the RT, but a discrete finite version.

To explore the FRAT mathematically let's consider a cyclic group Z_p denoted by $Z_p = (0, 1, ..., p-1)$ such that p is a prime number. Let the finite grid ${Z_p}^2$ be defined as the Cartesian product of $Z_p \times Z_p$. This finite grid has $(p+1)$ non trivial subgroups [79], given by:

$$L_{k,l} = \{(i,j) : j = (ki + l)(\mathrm{mod}p), i \in Z_p\}, \quad k < p \tag{4.1}$$

and

$$L_{p,l} = \{(l,j) : j \in Z_p\} \tag{4.2}$$

where each subgroup $L_{k,l}$, is the set of points that define a line on the lattice Z_p, k and l represent the slop and intercept of the line respectively. The Radon projection of the function f on the finite grid Z_p^2 is [79] then given by:

$$r_k[l] = FRAT_f(k,l) = \frac{1}{\sqrt{p}} \left(\sum_{(i,j) \in L_{k,l}} f[i,j] \right) \tag{4.3}$$

The FRAT is the basic building block for a number of transforms, including the FRIT and curvelets, more discussion of the theory, mathematics and applications of the FRAT can be found in [79]. It has been also been shown in this work that the Filtered Back Projection (FBP) provides a perfect inversion for the FRAT. Also, the algorithm for the FBP and FRAT are synonymous. Hence the same architecture can be used to implement both the forward and inverse transforms. The overall design flow of FRAT is presented in

Fig. 4.1 which is based on the standard FRAT pseudocode presented in [79]. It is clear form the flow chart that an image is divide into number of sub-blocks with size of p where p is a prime number. Then the FRAT operation is applied to each block separately. It takes the average of each row of the input block and replace it in the first row of the new FRAT block. The rest of the new block's rows depend on the value k, l where k and l are the slop and the intercept of the line, based on the value of k, l, it adds all the pixels of the line and find the averages and put them in the second, third and others row of the new FRAT block. For the final row of the new block it adds and finds the average all columns of the input block and replace in the last row of the new block. At the end the new FRAT block merges. The same procedure will be applied to all blocks.

4.3 Proposed Architectures for FRAT - Design and Evaluation

In this section, the proposed FRAT architecture is explored which is a serial I/O architecture with a parallel core that computes all $p + 1$ FRAT vectors simultaneously. The block diagram describe the proposed FRAT architecture is presented in Fig. 4.2. As it is clear for the diagram it is a serial input architecture. The advantage of a serial input architecture is that the FRAT block can be easily included into a sequence of image processing/compression steps such as the ridgelet or curvelet, without imposing any restrictions on the nature of the inputs. No clock cycles are wasted in buffering the whole input block, and the input section can be pipelined. The controller has $p + 1$ counters with size of 8-bits which generate the address and the read/write status of the output buffer (dual port RAMs). The address vector decoder generates the correct sequence of addresses accessed in the output buffer based on the values of *k,l*. Each accumulator reads the contents of the specified location from the dual port RAMs, adds the data from the input port, and writes it back to the deferent location of the RAM, since it is a dual port RAM, all operation happens within the same clock cycle. The input and core section of the design are completely pipelined. The output stage cannot be pipelined because the FRAT is an over-complete transform that yields $p(p+1)$ output points for p^2 input points. For maintaining design simplicity and due to power considerations, a non-pipelined serial data flow in the output section is adopted. The generated FRAT vectors will be stored by a number dual part RAM. In total, there are p independent dual port RAM buffers at

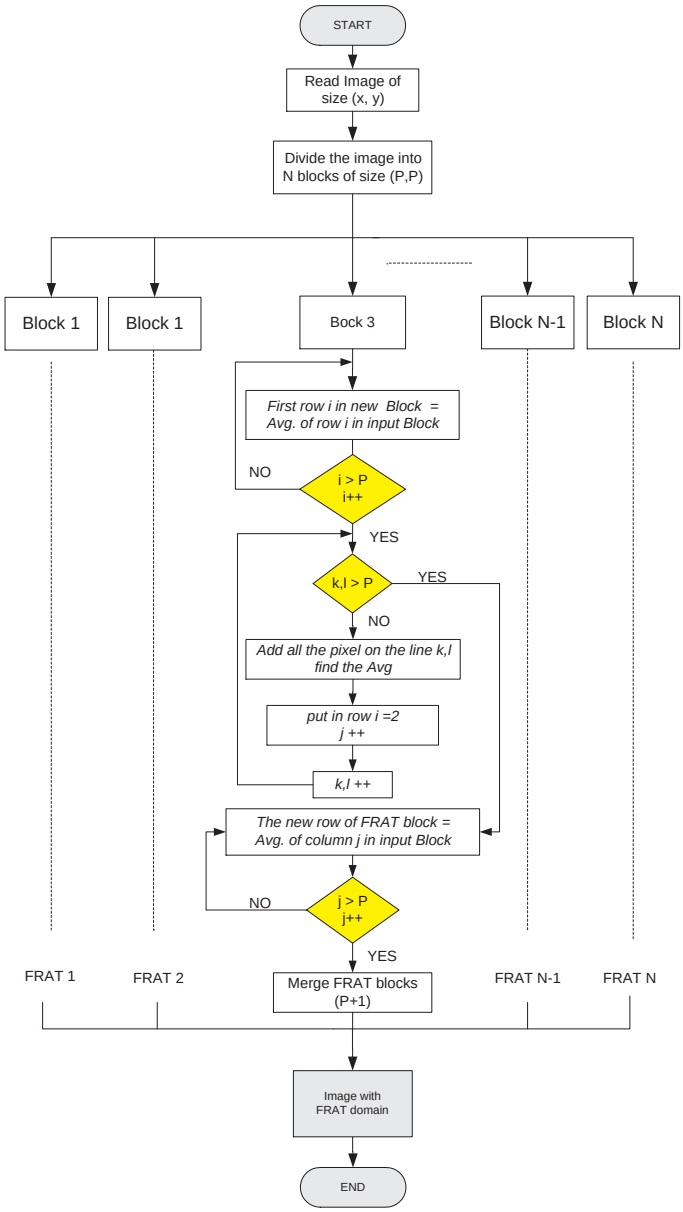


Figure 4.1: FRAT Flowchart

the output section, one for each FRAT vector. The architecture uses a distributed RAM (LUTS) that stores the address values for each FRAT vector rather than multiplexer or counter chains that have been used in previous designs. Multiplexers and arrays utilise deep logic, and also lie in the critical path of the design. This naturally reduces the maximum frequency obtained. The novelty of this architecture lies in the fact that a different approach has been taken to implement the FRAT efficiently, instead of just parallelising the standard FRAT pseudocode presented in [79]. Additionally, instead of using arrays to store the output coefficients, the design takes the advantages of hardware resources available on the new advanced FPGA and using dual ported RAMs which helps in reducing the

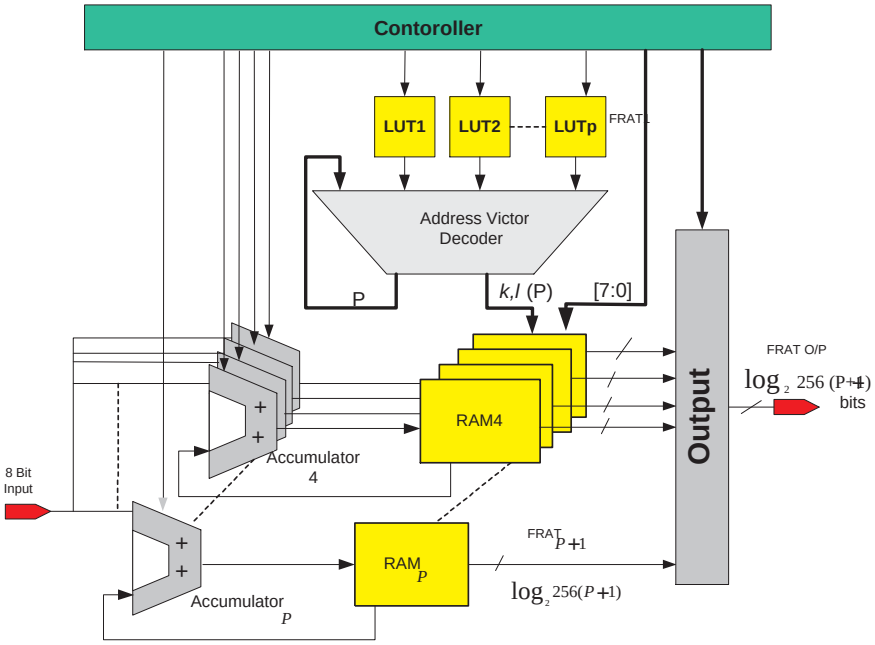


Figure 4.2: Proposed architecture for FRAT

area. This used of dual port RAMs also contributes to improved frequency performance when compared to previous designs due to multiple access within one clock cycle. At the end of $p2$ clock cycles, the output contains the image block in the transform domain. They are then ejected in a serial fashion from the output port. It ia worth mentioning that the output size for each FRAT is $p+1$ with pixel size of 8-bits ($\log_2 256(P+2)$).

4.3.1 FRAT Comparison with Existing Architectures

The Time Complexity (TC), Area Complexity (AC) and I/O types obtained for the proposed FRAT block are presented in Table 4.1.

Table 4.1: Comparison with existing architectures

	TC	AC	I/O
Proposed	$O(p^2)$	$O(2p^2)$	Serial
[17] Generic	$O(p^2)$	NA	Parallel
[17] Serial	$O(p^2(p+1))$	$O(2p^2)$	Serial
[15] Reference	$O(p^2(p+1))$	$O(2p^2)$	Serial
[15] Memoryless	$O(p(p+1))$	NA	Parallel
[14] Parallel	$O(p^2)$	$O(2p^2)$	Parallel
[16] Reference	$O(p^2(p+1))$	$O(2p^2+p)$	Serial

It can be seen from Table 4.1 that the proposed parallel architecture has the least time complexity compared to other parallel cores. Since the inputs and outputs are stored in

RAM and distributed RAM the area is the same other parallel cores. It is also worth mentioning that the type of I/O used also has an impact on the I/O power of the design. Serial architectures in general consume lesser I/O power when compared to parallel architectures.

4.4 FRAT FPGA Implementation

In this section the proposed FRAT IP core implementation on different FPGA platforms has been explored. In addition, evaluation of performance in comparison with other existing work are also presented. In order to verify the performance of the proposed architecture

Table 4.2: Performance matrices of FRAT core on different FPGA platforms

Platforms	Parameters	P = 7	P = 17	P = 31
Spartan-3L	Area (Slices)	312	520	1007
	Max Freq. (MHz)	101.2	96.06	90.82
	Dual P. RAM	176	468	832
	Registers	18	65	193
Virtex-2	Area (Slices)	244	756	1436
	Max Freq. (MHz)	180.83	111,98	103,62
	Dual P. RAM	176	936	1664
	Registers	20	56	193
Virtex-5	Area (Slices)	133	354	608
	Max Freq. (MHz)	175.16	166.19	154.12
	Dual P. RAM	96	252	392
	Registers	19	54	113

for FRAT has been prototyped on the Celoxica RC1000 [33]. In order to provide a fair results the IP cores developed are re-synthesised and implemented without any architectural modifications with advanced FPGA platform Xilinx LX110T (Virtex-5). Synthesis is also carried out on the low power FPGA Spartan3L 3s1500l-4 platform. A brief overview and features of Vitex-5 and Spartan-3 FPGA platforms used are provided in Appendix A. Various performance metrics like area occupied, maximum frequency, number of dual port RAMs and registers of the implementation results obtained for proposed FRAT is presented in Table 4.2. In order to provide a fair comparison with other existing work the design has also implemented with Xilinx Virtex-2. The results obtained have been compared with other existing work. It can be seen from table 4.3 that the implementation results from the proposed architecture outperforms other existing implementations results in terms of area and specially maximum frequency. This is because the proposed architec-

ture has parallel cores and using RAMs to store data rather than arrays which consume more hardware. It is worth noting that the reference and parallel in Table 4.3 represent the method of the design, reference means serial core and parallel means parallel cores.

Table 4.3: Comparison of performance metrics with existing FPGA implementations

Platform	P	Design	Area (Slices)	Freq. (MHz)
Virtex-2	7	Proposed	244	180.83
	17	Proposed	756	111,98
	31	Proposed	1436	103,62
Virtex-2	7	[15] Reference	159	100.13
	7	[15] Parallel	558	81.92
Virtex-2	7	[14]Parallel	245	96.46
	17	[14]Parallel	824	N/A
Virtex-2	7	[16]Parallel	198	112.86
	17	[16]Parallel	638	87.44
	31	[16]Parallel	1118	80.29

4.4.1 Applying FRAT on Image Data

In recent year FRAT has been used for many applications including medical imaging (tomography) [103]. The FRAT has been applied on medical images (human chest, lung and brain which are obtained from a CT scanner) for evaluation purposes which is shown in Fig. 4.3. It can be observed that the averaging effect of the FRAT and the block artifacts of the image in the transform domain become clearly visible as p increases. It is worth mentioning that the Peak Signal to Noise Ratio (PSNR) depends only on the accuracy required. The PSNR is used in image processing as a physical measure of the sensitivity of an imaging system. Image quality is excellent when PSNR is 32 dB and if PSNR is 20 dB the image is acceptable in term of quality. The truncation or rounding step that follows the FRAT determines the PSNR figures. The rounding or truncation process can easily be incorporated along with the wavelet block with no extra computational effort by suitably modifying the wavelet coefficients. However, to illustrate the effect of bit-width limitations on PSNR, reconstruction has been carried out on standard images stored at eight Bits Per Pixel (BPP) in the FRAT domain. The PSNR values of the reconstructed images have been presented in Table 4.4. It can be seen from Table that the PSNR of the reconstructed image image drops almost by 10dB when the block size increased from P=7 to P=31. This is because as p increases, the rounding error become more significant. Using

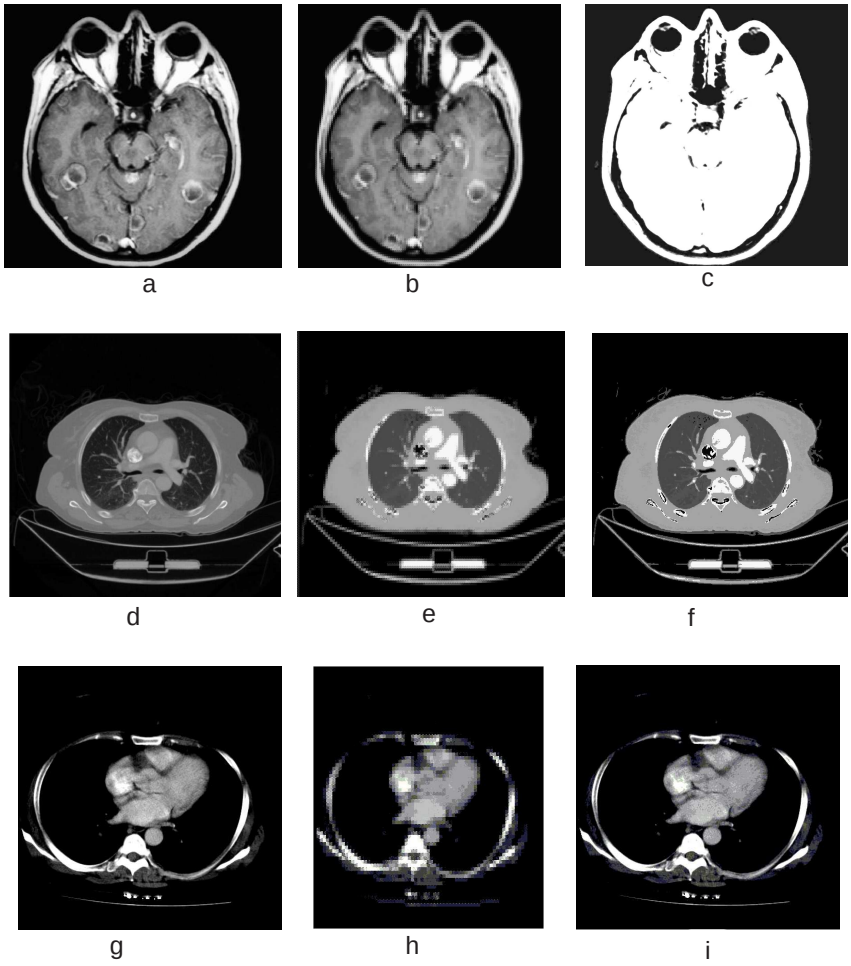


Figure 4.3: Spatial domain and transformed images (a) Spatial domain human brain (b) FRAT domain, $p = 7$ (c) Reconstructed image (d) Spatial domain human lung (e) FRAT domain, $p = 7$ (f) Reconstructed image (g) Spatial domain human chest (h) FRAT domain, $p = 7$ (i) Reconstructed image

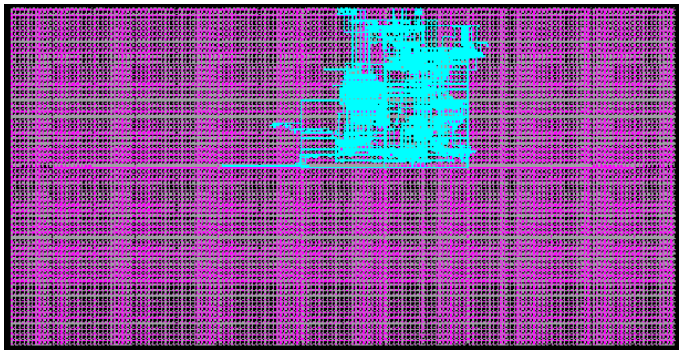
a driver with greater precision can reduce the rounding error. It also worth mentioning that when the block size is increasing, the processing time is also increasing.

4.4.2 Chip Level Details

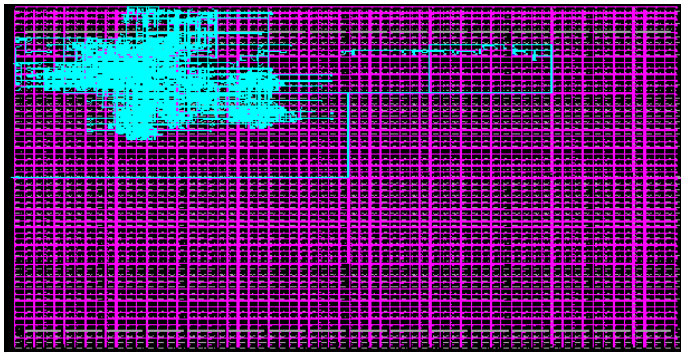
In addition to FPGA implementation, a careful manual PAR of critical nets and manual pin assignment for the designs has been performed using Xilinx Floorplanner [2]. The FPGA chip layout for FRAT post PAR when $P = 17$ is shown in Fig. 4.4.

Table 4.4: The PSNR analysis of reconstructed images from 8 bit FRAT images

Image Name	**Block** Size	**FRAT** PSNR (dB)	**FRAT Processing** Time (sec)
Brain	7	49.66	0.37
	13	39.23	0.62
	31	31,56	0.93
Lung	7	28.67	0.92
	13	21.92	1.21
	31	17.43	1.54
Chest	7	53.71	0.92
	13	50.32	0.97
	31	45.29	1.44



(a)



(b)

Figure 4.4: Chip layout of Virtex-5 and Spartant-3L for FRAT (a) Chip layout of Virtex-5 for FRAT when the block size (P) is 17 (b) Chip layout of Spartan-3L for FRAT when the block size (P) is 17

4.5 Finite Ridgelet Transform: A Brief Review

FRIT has gained attention recently due to superior compaction over wavelets due to its directional nature, resulting in better performance in applications such as compression and

denoising. These applications are computationally intensive, which require realtime acceleration process. Software implementation of the FRIT is usually based upon the standard pseudocode for the FRAT presented Appendix B followed by a suitable DWT block. The computational complexity of the FRAT and DWT blocks in the FRIT make hardware acceleration essential for fast computing and achieving real time processing. A straightforward translation of this pseudocode for hardware implementation is sub-optimal, as it does not exploit the parallelism capabilities of dedicated processors. Non-dyadic transform lengths and modulo operations in the FRAT sub-block necessitate algorithmic transformations to yield efficient hardware implementation. The DWT block is based on the HWT, which provides the best energy compaction (minimum entropy) when compared to other higher order wavelets. It is the aim of this work to present a novel and platform independent VLSI architecture for FRIT. Efficient FPGA implementation of the proposed architecture is also performed on different FPGA platforms for evaluation.

4.5.1 Mathematical Background of the Finite Ridgelet Transform

The FRAT has been described in details in the previous section. Additionally, a comprehensive mathematical review about the FRAT has also been provided in previous section. The final expression for the FRAT is reprinted here as a refresher:
The Radon projection of the function f on the finite grid Z_p^2 is given by:

$$r_k[l] = FRAT_f(k,l) = \frac{1}{\sqrt{p}} \left(\sum_{(i,j)\in L_{k,l}} f[i,j] \right) \tag{4.4}$$

4.5.2 Haar Wavelet Transform

There are many popular discrete wavelets which are used in different applications. Among these wavelets, HWT is one of the simplest wavelet transforms. Although its simplicity makes it unsuitable for a number of applications, particularly due to the fact that it is not continuous and hence not differentiable, it is highly suitable for constructing the FRIT due to the following reasons. Firstly, the HWT is very simple in structure and is less expensive to compute compared to other wavelets. Secondly, due to the averaging characteristics of the FRIT, it has been observed that the HWT displays superior performance in energy compaction when compared to other wavelets. HWT has been explored in details in

Chapter 3.

4.5.3 Building the FRIT from FRAT and DWT

FRIT is obtained by performing the HWT on each FRAT projection sequence with fixe value of k. The continuous ridgelet transform of a bivariate function $f(x)$ is given by [48]:

$$RT = \int_{R^2} \psi_{a,b,\theta}(x) f(x) dx \tag{4.5}$$

However, the use of digital images necessitates the development of suitable variations of the ridgelets to deal with images in the digital domain. The process of FRIT is represented in Fig.4.5.

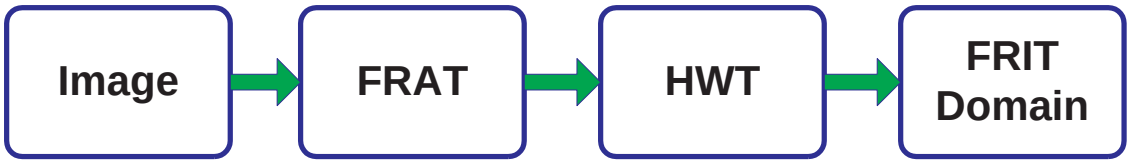


Figure 4.5: Finite ridgelet transform obtained by performing HWT on the FRAT vectors.

Mathematically, this is given by [46]:

$$FRIT_{i,i+1}(k,l) = \lfloor FRAT_{i,i+1}(k,l) \times H_2 \rfloor_2 \forall i = 1,3,...p \tag{4.6}$$

The directional attributes of these higher dimensional generalisations of wavelets make them ideal for a number of applications such as alternate image representation, compression, denoising, etc.

4.6 Proposed Architectures for FRIT

In this section, the design flow used to generate the optimised architecture for FRIT using the novel FRAT block which has been presented in section 4.3 and novel HWT block which has been explored in Chapter 3 is presented. The generated FRAT vectors will be stored in dual part RAMs. In totally, there are p independent dual port RAM buffers at the output section, one for each FRAT vector. The block diagram description of the FRIT architecture with parallel core is shown in Fig.4.6. The HWT sub-block is fully pipelined,

and is efficiently implemented using accumulate shift and accumulate subtract sub-unit circuits that perform the combined task of computing the sum and differences, as well as scaling the image in the wavelet domain. It can be seen from Fig. 4.6 that it consist two part, FRAT part which is the same FRAT which has been explored earlier in this chapter. The other part is HWT, HWT is the simplest wavelet and has a better entropy measures which indicate that it has a better performance than other complex wavelets for performing the FRIT. The outputs of the FRAT domain with size $p+1$ is fed as inputs to the HWT then the HWT operation (averaging and differencing) is applied to obtain the FRIT outputs. The controller generate the address and the read/write status of the dual port RAMs, it also select the outputs sequences of FRAT to HWT. It is worth mentioning that the HWT is based on Haar Wavelet Transform Factorisation Method 1 (HWTFM1) which has been explored in Chapter 3.

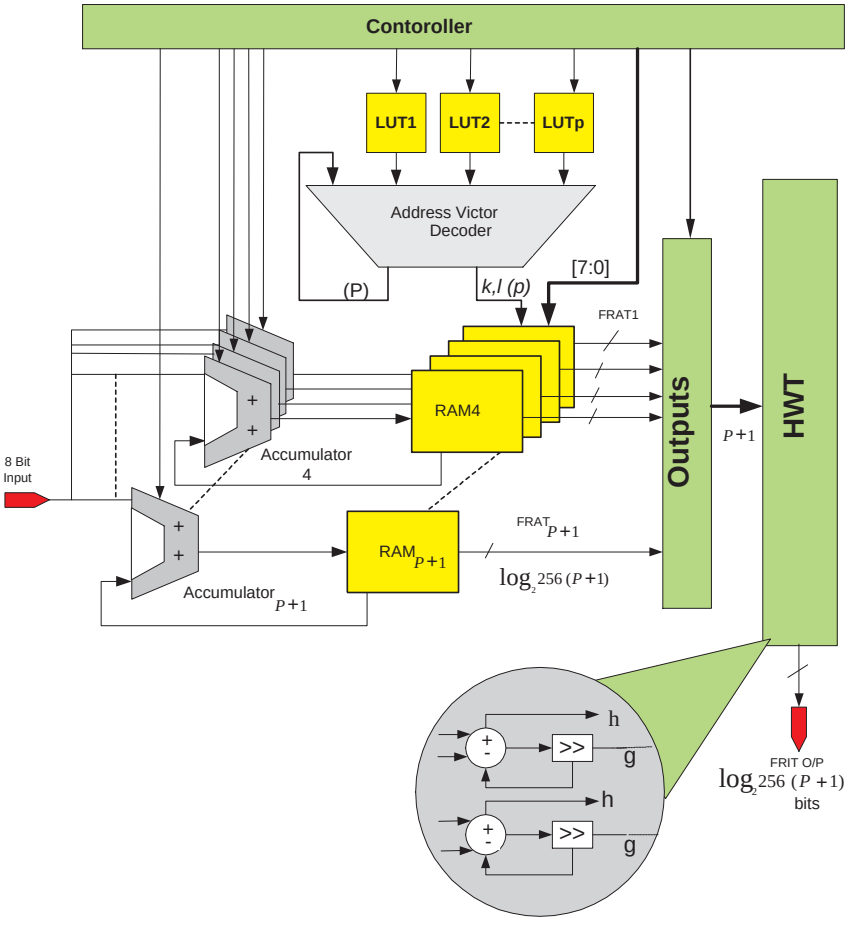


Figure 4.6: Proposed architecture for FRIT

4.6.1 Results and Analysis

In order to evaluate the performance of the proposed FRIT, the designs parameters have been compared with other existing architectures. Additionally, the proposed FRIT has also been implemented on different FPGA platforms, the obtained results have been compared with other existing FPGA implementations in place. The results comparison is present in the following subsections.

Comparison with Existing Architectures

Design parameters such as TC, AC and I/O type of the proposed design and other existing architectures are presented in Table 4.5. It can be seen that the proposed architecture has a competitive area complexity figure. It can also be seen from Table 4.5 that the proposed architecture has the least time complexity compared to some other IP cores presented in Table 4.5. It is also worth mentioning that the impact of I/Os are very important in FPGA power consumption. Serial architectures in general consume less power compared to parallel architectures.

Table 4.5: Comparison of design parameters of FRIT with existing architectures

Design	TC	AC
Proposed Architecture	$O(p^2)$	$O(log_2 p + 2p^2)$
[17] Generic	$O(p^2)$	NA
[17] Serial	$O(p^2(p+1))$	$O(2p^2)$
[15] Reference	$O(p^2(p+1))$	$O(2p^2)$
[15] Memoryless	$O(p(p+1))$	NA
[14] Parallel	$O(p^2)$	$O(2p^2)$
[16] Parallel	$O(p^2(p+1))$	$O(2p^2+p)$

4.6.2 Image Data

FRIT has been applied on some medical images for evaluation and the outcome is shown in Fig. 4.7. The truncation or rounding step that follows the HWT sub-block determines the PSNR figures. The level of precision required is a choice best left to the end user. Hence the IP cores that have been developed operate at full precision. The PSNR and the processing time for FRIT is shown in Table 4.6. It is worth mentioning that when p is increasing the PSNR is not changing significantly, but the time processing is increasing by 50 % when p changes from 7 to 13.

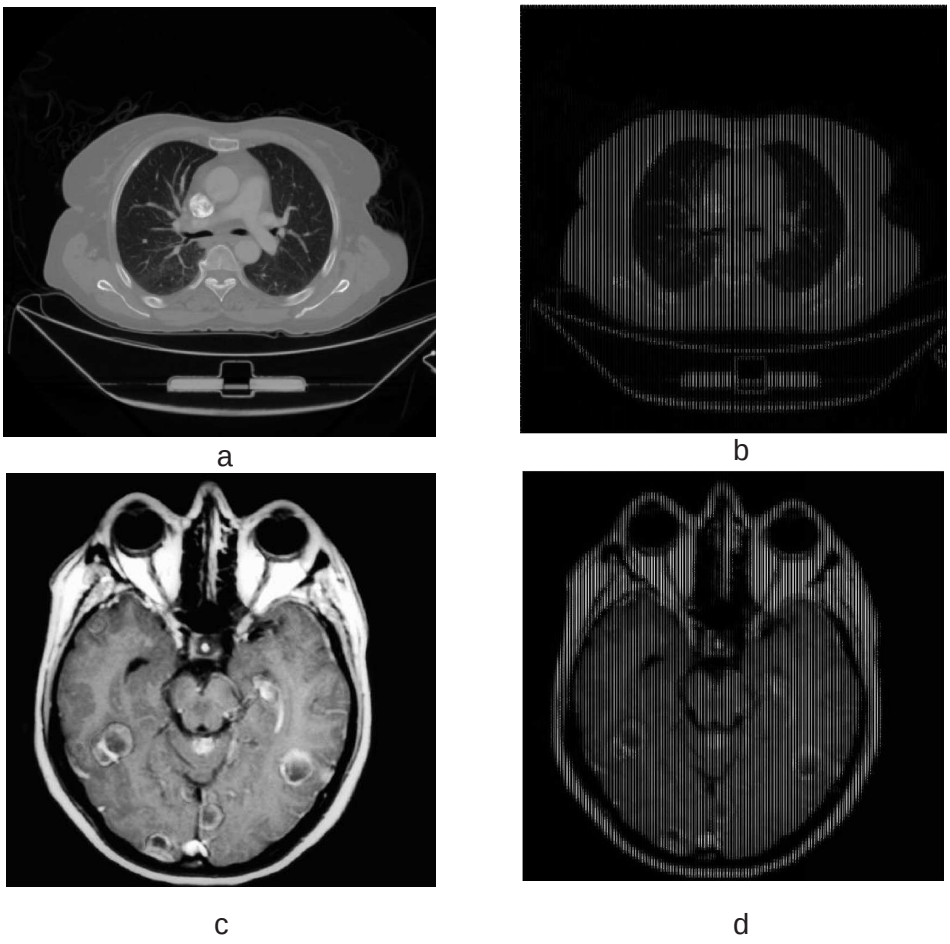


Figure 4.7: Spatial domain and transformed images (a) Spatial domain human lung (b) FRIT domain, $p = 7$ (c) Spatial domain human brain (d) FRIT domain, $p = 7$

Table 4.6: The PSNR and timing analysis of 8 bit FRIT images

Image Name	**Block** Size	**FRIT** PSNR (dB)	**FRIT Processing** Time (sec)
Lung	7	4.55	23.06
	13	3.67	16.71
	31	2.81	9.41
Brain	7	5.52	55.33
	13	4.12	34.91
	31	3.21	27.12
Chest	7	8.13	47.42
	13	6.62	24.97
	31	6.29	10.44

4.6.3 FRIT Host-FPGA system

The Host application provides a user interface with pull down menus to interactively select the image to be transformed. Based on the requirements of the system architect, the

block size required for the FRIT can be selected. The corresponding FPGA configuration bitstream is loaded on the FPGA initiating the transformation process. Next, the Host application sends the data to flash memory RC FPGA boards. On completion the Host sends a signal to the FPGA and releases control over the flash memory. The FPGA takes control over the flash memory, and reads the image from the flash. Each image block is transformed to the FRIT domain stored back into flash memory. After the entire image is processed, control over the flash memory is released back to the Host. The Host reads the transform image from flash memory and displayed in the user interface. A snapshot of the host-FPGA system is shown in Fig. 4.8. It is worth mentioning that the maximum size flash memory on the RC10 board is 16 MB.

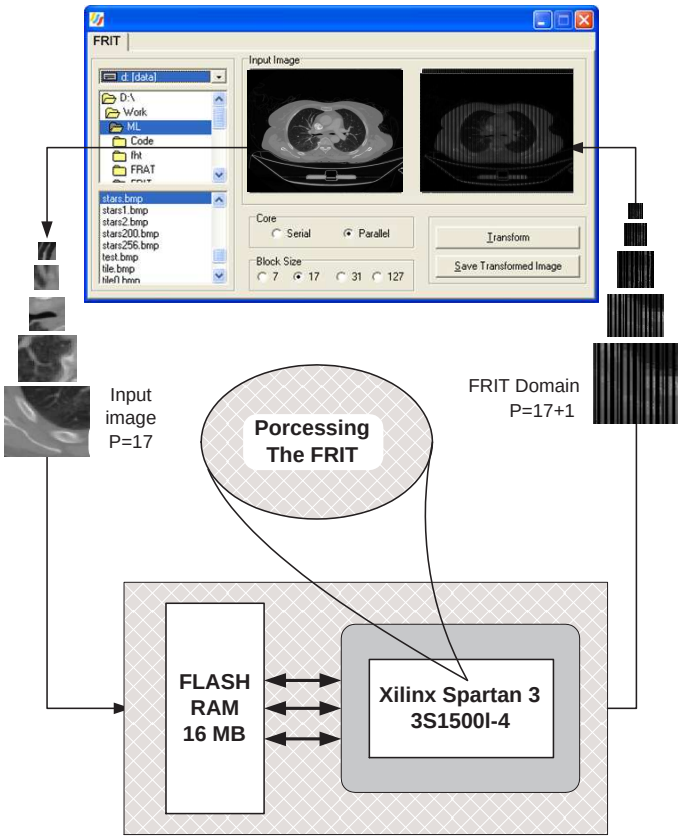


Figure 4.8: Host application for FRIT

4.7 FRIT FPGA Implementation

In this section, various details of FRIT implementation on different FPGA platforms has been explored. In addition, evaluation of performance in comparison with existing work,

chip diagrams are also presented. In order to provide to compare with other existing work FRIT has been prototyped on the Celoxica RC10 [33]. The FRIT IP core developed are also re-synthesised and implemented without any architectural modifications with Virtex-5 (LX110T) platform to verify the performances. Synthesis is also carried out on the low power FPGA Spartan 3 to perform a critical comparison of power dissipation on a low-power FPGA platform, more details about these platforms can be found in Appendix A. The implementation results obtained for proposed FRIT is presented in Table 4.7. The

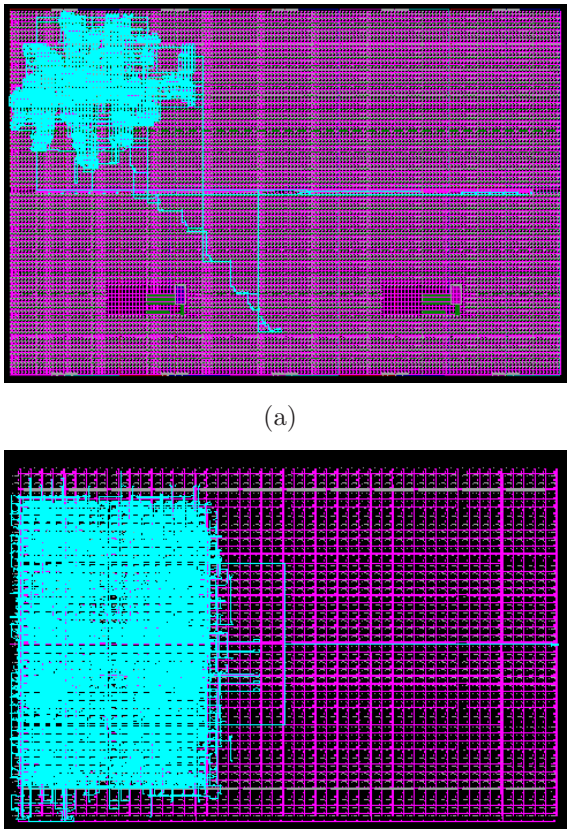
Table 4.7: Comparison of performance metrics with existing FPGA implementations

Design	P	Platform	Area (Slices)	Freq. (MHz)
Proposed	7	Virtex-5	186	173.55
Proposed	17	Virtex-5	502	166.23
Proposed	7	Spartan-3L	298	85.5
Proposed	17	Spartan-3L	1,110	48.6
Proposed	7	Virtex-E	309	72.63
Proposed	17	Virtex-E	1,112	51.56
[17] Generic	17	Virtex-E	n/a	33.00
[17]Serial	17	Virtex-E	828	18.00
[14]Parallel	7	Spartan-3L	312	76.3
[14]Parallel	17	Spartan-3L	1,124	40.23
[14]Parallel	7	Virtex-E	312	70.55
[14]Parallel	17	Virtex-E	1,176	40.23
[16]Reference	7	Virtex 2	476	101.00
[16]Reference	17	Virtex 2	911	32.00

implementation results from the proposed architecture of FRIT outperformed the other existing implementations results in terms of area and maximum frequency. This is because the proposed architecture has a parallel FRAT core and a DA based HWT. It can be seen from Table 4.7 the FRIT core is implemented and tested with p= 7 and 17. The advance FPGA (virtex-5) has the highest frequency which is almost 50% faster than other FPGAs and utilizes almost 50% less logic (slices), it is because of the structure of new CLBs in Virtex-5. It can be also seen from the Table 4.7 that proposed FRIT (with parallel cores and DA based HWT) improved the frequency by 11% compare to other recent parallel cores implementations and reduces the logic resources used almost by 8%. It is worth mentioning that Generic, Reference and Parallel in Table 4.7 represent the methodologies, Parallel means parallel core FRAT and Reference means serial FRAT.

4.7.1 FPGA Chip Level Details

In addition to FPGA implementation, a careful manual PAR of critical nets and manual pin assignment for the designs has been performed using Xilinx Floor planner [2]. The FPGA chip layout for FRIT post place and route when $P = 31$ is shown in Fig. 4.9.



(a)

(b)

Figure 4.9: Chip layout of Virtex-5 and Spartant-3L for FRIT (a) Chip layout of Virtex-5 for FRIT when the block size (P) is 31 (b) Chip layout of Spartan-3L for FRIT when the block size (P) is 31

4.8 Conclusions

In this Chapter, an architecture with a parallel core for FRAT which is suitable for FRIT has been proposed. The outputs of proposed architecture are a non-pipelined serial data flow and They are stored by a number dual part RAMs. In total, there are p independent dual port RAM buffers at the output section, one for each FRAT vector. The architecture uses a distributed RAMs (LUTS) to stores the address values for each FRAT vector rather

than multiplexer or counter chains that utilise deep logic. This naturally helped to reduces the maximum frequency and area. The used of dual port RAMs have also contributed to improved frequency performance, compared to previous designs due to multiple access within one clock cycle. At the end of $p2$ clock cycles, the output contains the image block in FRAT domain. They are then ejected in a serial fashion from the output port. The proposed FRAT architecture has a core latency of $O(p^2)$ and AC $O(2P^2)$. It ia worth mentioning that the output bus for each FRAT is $p+1$ with word-length of 8-bits ($\log_2 256(P+2)$). The evaluation of the implementation results has shown that the proposed architecture outperformed other existing results in place.

The proposed FRAT architecture is combined with HWTFM2 which has been explored in Chapter 3 to present a high performance, power aware and optimised architecture for FRIT core. The proposed FRIT architecture performance has been evaluated and the results have outperformed some other existing architecture in place. It is worth mentioning that since FRAT is mean building block of FRIT the TA is the same as FRAT and the AC is changing $O(log_2 p + 2P^2)$ because of HWT. Both proposed architectures have been implemented on FPGA using Handel-C to gain maximum benefit in performance from the target hardware by using its parallel constructs. In order to evaluate the designs have been implemented and tested with p= 7 and 17 on low power and advanced FPGAs. The advance FPGA (virtex-5) has the highest frequency which is almost 50% faster than other FPGAs and utilizes almost 50% less logic (slices), because of the structure of new CLBs in Virtex-5. The evaluation of both architectures have shown that the obtained implementations results outperformed other existing results in place by almost 10% in terms of frequency and area.

The proposed architectures are also applied on image data (256×256). It has been observed that the FRAT PSNR of the reconstructed image image drops almost by 10dB when the block size increased from P=7 to P=31. This is because as p increases, the rounding error become more significant. On the other hand there is not a significant change in FRIT PSNR when P is changed from 7 to 31, but the time processing is increased almost by 50% when P is changed from 7 to 31. To summarise, architectural level optimisation techniques and FPGA implementation of FRAT and FRIT have been discussed in this chapter. In the next chapter, an efficient FPGA implementation of cyclic convolution using systolic architecture is discussed.

Chapter 5

Efficient FPGA Implementation of Cyclic Convolution using Systolic Design

5.1 Introduction

In digital signal processing, the design of fast and computationally efficient algorithms has been a major focus of research activity. Systolic designs represent an attractive architectural paradigm for efficient VLSI and Field Programmable Gate Arrays (FPGA) implementation of computation-intensive digital signal processing applications supported by the features like simplicity, regularity, and modularity of structure. Moreover, they also possess significant potential to yield a high-throughput rate by exploiting high level of concurrency using pipelining, parallel processing, or both.

In recent year several attempts have been made for implementation of signal processing IP cores like the Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT), Discrete Sine Transform (DST) and Discrete Hartley Transform (DHT) in systolic hardware through cyclic convolutional formulation [18,104–109] due to its remarkable advantage over the others, particularly for efficient input/output and data transfer operations. Moreover, the convolutional representation of all these transforms is found to be more suitable for hardware efficient high throughput memory based systolic realisation [106–108]. Significant work has been done to decompose the sinusoidal transforms into multiple shorter convolutions, and to implement them in parallel in order to design systolic structures with

lower area and time complexity [18, 105–109].

In this chapter, two architectures for cyclic convolution using optimal short length algorithm have been developed by applying principles of parallelism, pipelining and systolisation as appropriate. The proposed designs have been implemented on different FPGA platforms and the obtained results have been compared with other existing work. The presented architecture has been implementation using a power aware design flow and complete design space exploration. Power modelling details for one of the proposed architecture are presented in Chapter 6. It is worth mentioning that this work is a collaborative work with Pramod Kumar Meher School of Computer Engineering, Nanyang Technological University, Singapore. The first architecture and mathematical background in this chapter is presented by Meher. The second proposed architecture, FPGA implementation and result analysis for both architectures are explored by the author.

The rest of this chapter is organised as follows. Brief introduction into the principles and mathematical basis for cyclic convolution are presented in Section 5.2. The proposed architectures for cyclic convolution are explored in Section 5.3. The architectural results and FPGA implementation results are presented in Section 5.4. Concluding remarks are provided in Section 5.5.

5.2 Mathematical Background of Cyclic Convolution

In this section, the general mathematical formulation for cyclic convolution with block cyclic convolution algorithm have been explored. The general formula for cyclic convolution can be expressed as [110]:

$$y(n) \sum_{k=0}^{N-1} x(k)h((n-k) \bmod N) \; n = 0, 1....., N-1 \tag{5.1}$$

where x denoting the input sequence, h denoting the kernel, and y denoting the output sequence. It is understood that, in general, each of the three sequences may be complex. In this case the circular convolution is accomplished by forming the product as [111]:

$$Y_N = H_N.X_N \tag{5.2}$$

where Y= [y_0 y_1 y_2 y_3,....y_{N-1}] X=[x_0 x_1 x_2 x_3,....x_{N-1}] and H is a circulate matrix of size $N \times N$ which is generated by $h(N)$ and given by [111]

$$H = \begin{bmatrix} h_0 & h_{N-1} & h_{N-2} & . & . & . & . & h_1 \\ h_1 & h_0 & h_{N-1} & . & . & . & . & h_2 \\ h_2 & h_1 & h_0 & . & . & . & . & h_0 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ h_{N-2} & h_{N-3} & h_{N-4} & . & . & . & . & h_{N-1} \\ h_{N-1} & h_{N-2} & h_{N-3} & . & . & . & . & h_0 \end{bmatrix} \tag{5.3}$$

where the convolution length N is a composite number and in matrix form the cyclic convolution can be written as [111]:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ . \\ . \\ . \\ . \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} h_0 & h_{N-1} & h_{N-2} & . & . & . & . & h_1 \\ h_1 & h_0 & h_{N-1} & . & . & . & . & h_2 \\ h_2 & h_1 & h_0 & . & . & . & . & h_0 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ h_{N-2} & h_{N-3} & h_{N-4} & . & . & . & . & h_{N-1} \\ h_{N-1} & h_{N-2} & h_{N-3} & . & . & . & . & h_0 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ . \\ . \\ . \\ . \\ x_{N-1} \end{bmatrix} \tag{5.4}$$

5.2.1 Block Cyclic Convolution using Optimal Short Length Algorithm

Block cyclic convolution was first presented in [110], when the block cyclic convolution length is the element of composite number, it could be possible to compute the convolution output in a computationally efficient way using the optimal short-length cyclic convolution algorithms [111] and the mathematical block cyclic constellation can be presented as [110]:

$$Y_i = \sum_{j=0}^{M-1} C_{ij}.P_j, \ 0 \leq i \leq M-1 \tag{5.5}$$

$$where\ P_i = Q_i.S_i,\ 0 \leq i \leq M-1 \tag{5.6}$$

$$Q_i = \sum_{j=0}^{M-1} A_{ij}.H_j,\ 0 \leq i \leq M-1 \tag{5.7}$$

$$S_i = \sum_{j=0}^{M-1} B_{ij}.X_j,\ 0 \leq i \leq M-1 \tag{5.8}$$

Since we are using optimal short-length cyclic convolution the elements of A, B and C matrices are mostly very small integers [111], the matrix-vector products with these matrices can be preformed by repetitive addition/subtraction operations. In most practical situations, one of the input sequences is known as priori and remains fixed. The corresponding products of Eq.5.5, in such cases, may be pre-computed and stored. In term of hardware concern the most computation intensive part of this algorithm is the the Equ.5.6 which is implemented using systolic array design. In this work, the concentration is only on efficient hardware implementation of Eq. 5.6.

5.3 Proposed Systolic Architectures for Block Cyclic Convolution Algorithm

In this section, two architectures based on systolic array using parallelism and pipelining are proposed to implement the matrix vector products given by Eq.5.6. First architecture is based on pipelining the second architecture based on parallelism.

5.3.1 Proposed Pipelining Architecture

The proposed pipelining architecture is shown in Fig.5.1, which is a linear systolic array with pipelining process and consists of N Processing Elements (PEs) where N is the vector size.

The input sequences $(q_0 q_1 ... q_{N-1})$ and $(S_0 S_1 ... S_{N-1})$ are loaded in each PE and are staggered by one clock cycle with respect to its preceding PE to maintain the data dependence requirement. The Dependence Graph (DG) for the input sequences and its flow for the proposed architecture is presented in [110]. Inputs $(S_0 S_1 ... S_{N-1})$ remain stored in respective PEs for N cycles, while $(q_0 q_1 ... q_{N-1})$ gets shifted across the systolic array during each

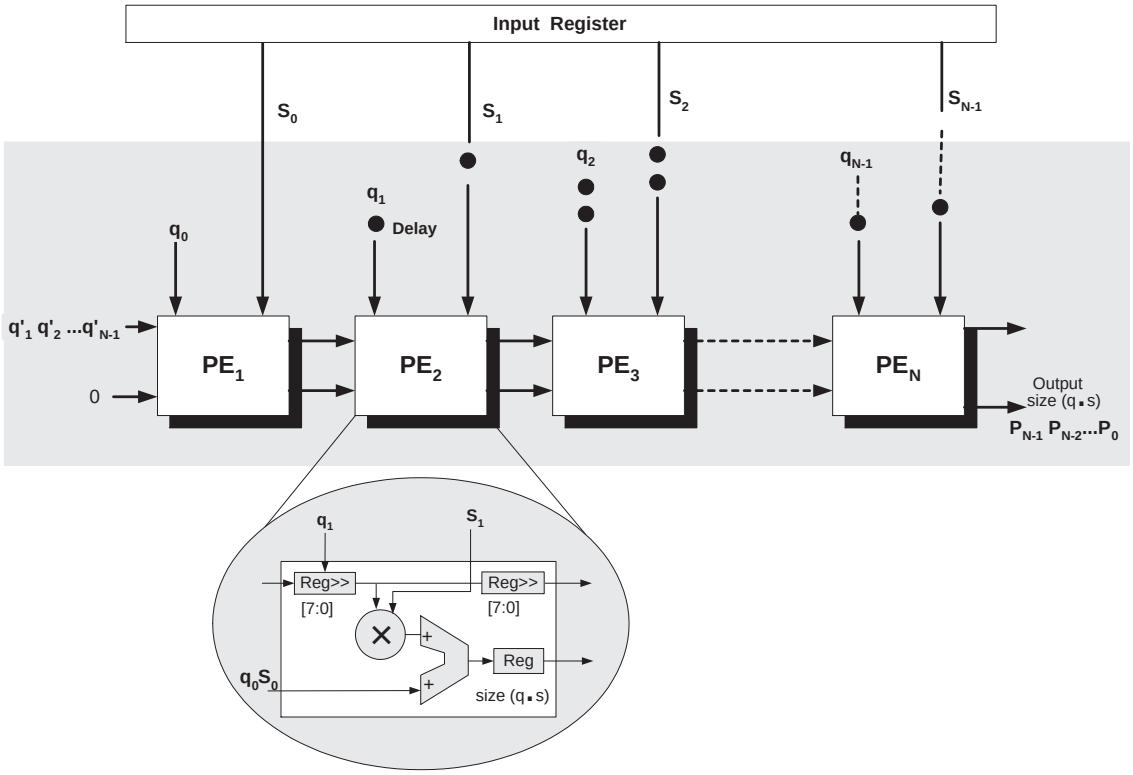


Figure 5.1: Proposed pipelined systolic architecture for block cyclic convolution

cycle. Input $(q'_1 q'_2 ... q'_{N-1})$ are fed one by one in each clock cycle to PE_1 a shown in Fig.5.1. The first convolved output is obtained from the structure after a latency of N cycles. The remaining outputs are obtained in every clock cycle after the N cycles. The final output is obtained after $(N-1)$ clock cycle. The processing operation of each clock cycle for proposed pipelined architecture when $(N=4)$ is shown in Fig. 5.2. The function of each PE is also shown in Fig. 5.1 which is consist of multiplier, adder, shift register and output registers which helps the flow of the pipelining. It is worth noting that the size of input registers are 8-bits and the output registers for each PE are the $q \times s$. In the first clock cycle it takes $S_0 \times q_0 + 0$ and store the result in a register to be ready for next PE. In the first clock cycle it also shift the input q'_{N-1} one step to the right. The same procedure is applied for all PEs until the last convolved output is obtained.

5.3.2 Proposed Parallel Architecture

The proposed systolic architecture for block cyclic convolution with parallel inputs is shown in Fig. 5.3. In this architecture all inputs $(q_0 q_1 ... q_{N-1})$ and $(S_0 S_1 ... S_{N-1})$ are fed in parallel. The function of each PE stay the same as the first architecture as it has been

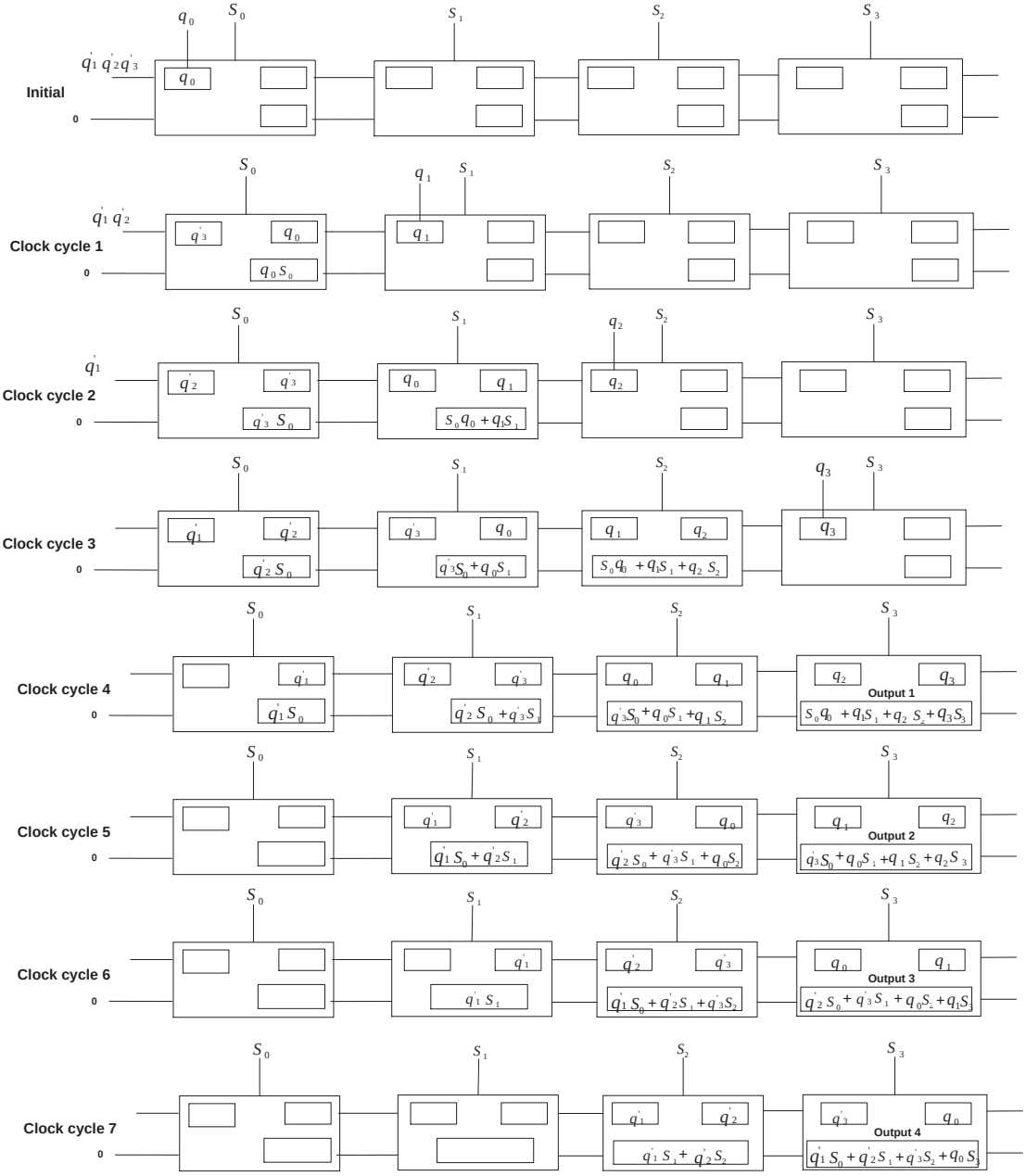


Figure 5.2: Data flow for proposed pipelining architecture when N is 4

outline in Fig.5.3, the only deference in this architecture is that all resources are used in each PE. It is clear from PE function in Fig. 5.3 that $(q_0 q_1 ... q_{N-1})$ are fed parallel in $N/2$ PEs which will be shifted every clock cycle to next PEs. The operation of the proposed architure for each clock cycle is presented in Fig. 5.4. In this case parallelism helps to reduce the area complexity rather than the time complexity. As it ia clear form Fig. 5.4 that after $4^t h$ the first convolved output is available and after $N-1$ clock cycle the last output is available which is the same time as first architecture. On the other the inputs

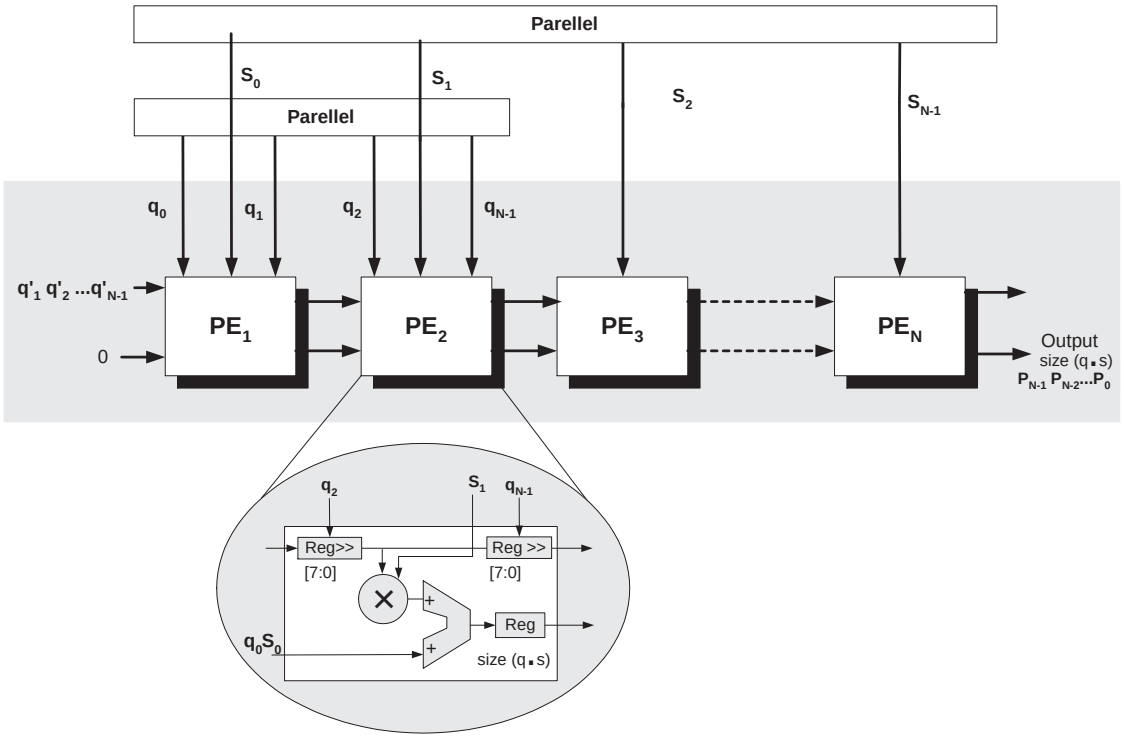


Figure 5.3: Proposed parallel systolic architecture for block cyclic convolution

are fed in parallel and there is no need for delay registers to hold $(q_0 q_1 ... q_{N-1})$ as it is in the first architecture.

5.4 Result and Analysis for Proposed Architectures

In order to evaluate the performance of the proposed architectures, the designs parameters have been explored and compared with other existing architectures. Additionally, the proposed architectures has also implemented on different FPGA platforms, the results have been compared with other existing FPGA implementations. The results comparison is present in the following subsections.

5.4.1 Architectural Results

The architecture comparison results of the design parameters with other existing architectures is presented in Table 5.1. It is clear from Table 5.1 that the TC of both architectures are the same and second architecture consume less registers. It is worth mentioning that

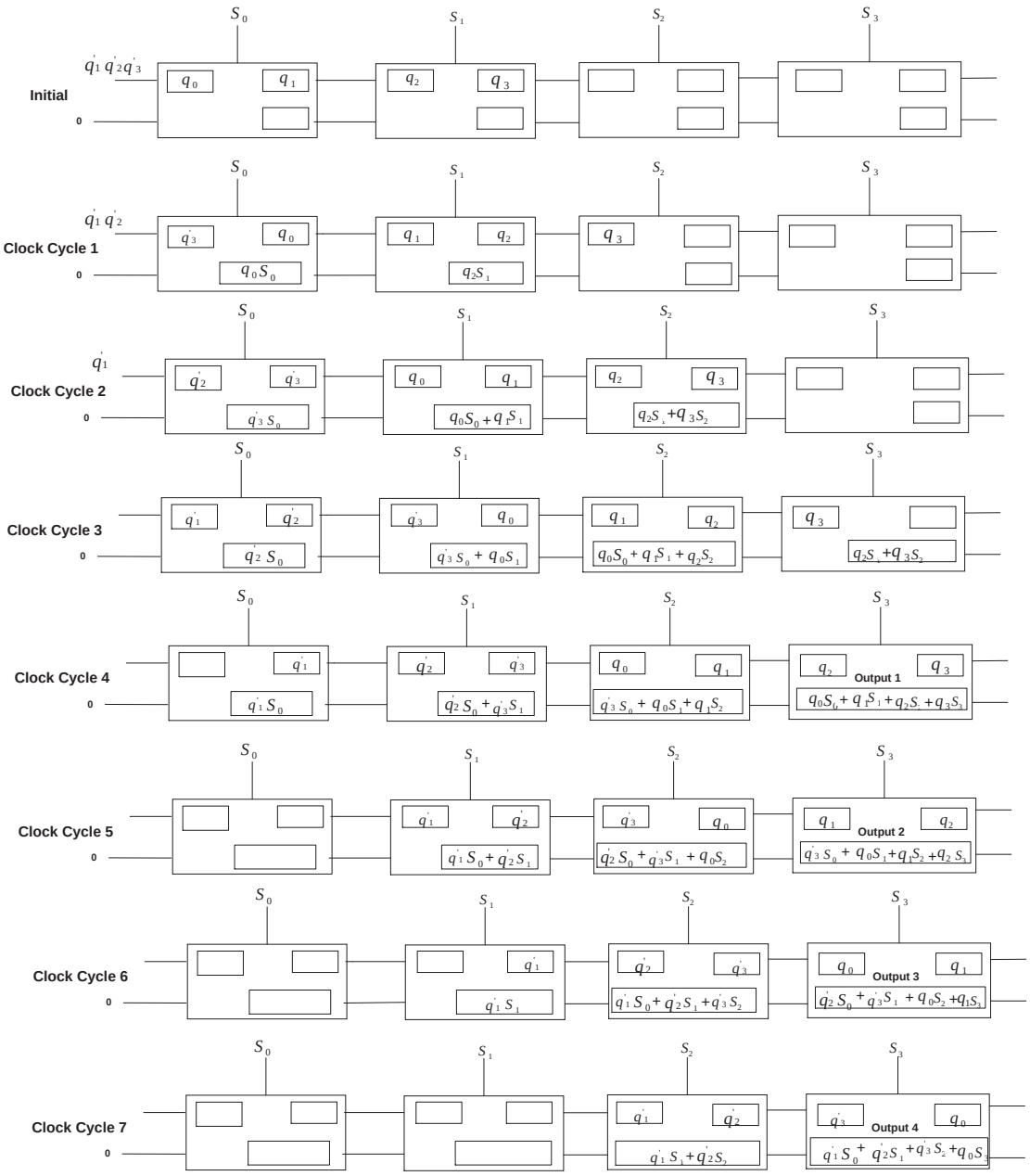


Figure 5.4: Data flow for proposed parallel architecture when N is 4

the proposed architectures show a significant improvement in terms of Time Complexity (TC) and number of registers compare to the other existing architecture in place. It is also worth noting that the in second proposed architecture the inputs $(q_0 q_1 ... q_{N-1})$ and $(S_0 S_1 ... S_{N-1})$ are fed in parallel which helps to reduce the registers by 42.8% compared to the first proposed architecture. This is because in the second architecture there is no need for delay registers to hold the value of $(q_1 ... q_{N-1})$.

Since power is one of the main concern of the design, It is also worth mentioning that the

Table 5.1: Comparison of the design parameters with other existing architectures (where N is vector size)

Design	TC	Multipliers	Registers	I/Os
Proposed 1	$(2N-1)$	(N)	$((2N-1)+\sum_{i=1}^{N} N_i - 1)$	Serial
Proposed 2	$(2N-1)$	(N)	$(2N-1)$	Perallel
[104]	$(2N)$	(N)	$(5N)$	N/A
[18]	$(N/2+2)$	(N)	$(4N-4)$	Serial
[110]	$(N/4+4)$	$(9N/8+4)$	$(5N-12)$	Serial

type of I/O has an important impact on power dissipation of the design. Serial architectures in general consume less I/O power when compared to parallel architectures. The proposed architecture can be modified even more by using a decoder for inputs $(q_0 q_1 ... q_{N-1})$ to reduce the inputs.

5.4.2 FPGA Implementation

In this section, FPGA implementation of the proposed pipelined cyclic convolution on different platforms has been explored. In order to verify the performance of the proposed architecture the design has been prototyped on the Celoxica RC1000 [33]. For further evaluation the design is re-synthesised and implemented without any architectural modifications with advance Xilinx FPGA Xc5vlx220(Virtex-5) platform. Synthesis is also carried out on the low power FPGA Spartan-3 3s1500l-4 platform to perform a critical comparison of power dissipation on a low power FPGA platform.

Table 5.2: FPGA implementations results for the proposed cyclic convolution architecture

FPGA Platforms	N	W	Area (Slices)	I/Os	DSP	Freq. (MHz)
Virtex-E	4	8	296	89	N/A	81.55
Virtex-E	16	8	1211	281	N/A	76.23
Virtex-E	32	8	1,695	403	N/A	68.9
Spartan-3L	4	8	60	90	N/A	126.50
Spartan-3L	16	8	253	282	N/A	120.5
Spartan-3L	32	8	512	404	N/A	117.50
Virtex-5	4	8	35	90	4	298.95
Virtex-5	16	8	117	282	16	290.95
Virtex-5	32	8	268	404	32	152.89

The implementation results obtained for proposed architecture is presented in Table 5.2,

where N is the vector size and W is the word-length. It is clear from Table 5.2 that there is a significant difference in terms of frequency and area occupied (slices) between each FPGA platforms. For instance Virtex5 uses less (30%) slices compared to Virtex-E, in other words the variation of the area in each platform depends on the technology used which can be further improved if available resources on these platforms are used. In Virtex4, CLBs contain four slices, each slice has two LUTs and two flip-flops. In the latest FPGA platform (Virtex5), CLBs have two slices which contain four LUTs and four flip-flop compared to previous platforms. Virtex-E platform have two slices per CLB which occupies more slices by the design. The slice and LUT utilisation on different FPGA platforms are presented in Fig. 5.5 and Fig. 5.6 respectively. It is clearly visible from Fig. 5.5 and Fig. 5.6 that latest FPGA (Virtex-5) uses less logic than Spartan-3 with same N value. It can be seen from Fig. 5.5 and Fig. 5.6 that the deference between Virtex-5 and Spartan-3 is not clear when N is 4, this is because of large scales of the graph, otherwise the deference between them are almost 40% for all value of N including N=4.

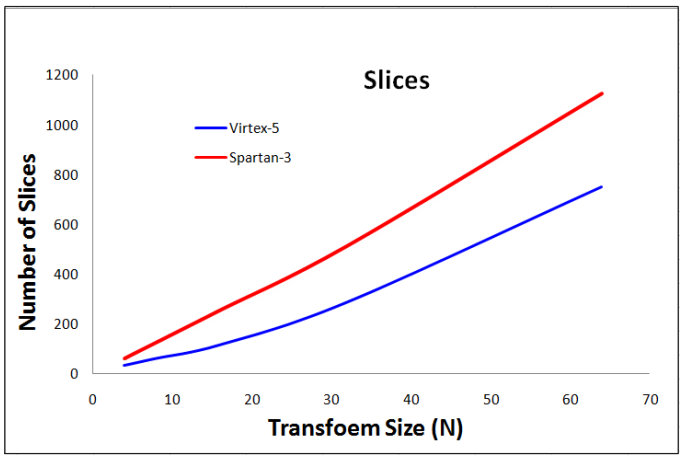


Figure 5.5: Slice utilisation with different transform sizes on different FPGA platforms

In order to obtain a comparison the proposed design has been implemented on Virtex-4 (XC4VS X35) and the results have been presented in Table. 5.3. It is clear from Table. 5.3 that the results for proposed design has significantly improved in terms of Max. frequency and some other elements compare to [112]. Since we are implementing Eq.5.6 which is a matrix vector multiplication, it is worth mentioning that the obtained results has been compared with existing implementation of FPGA matrix vector multiplication results rather than cyclic convolution results.

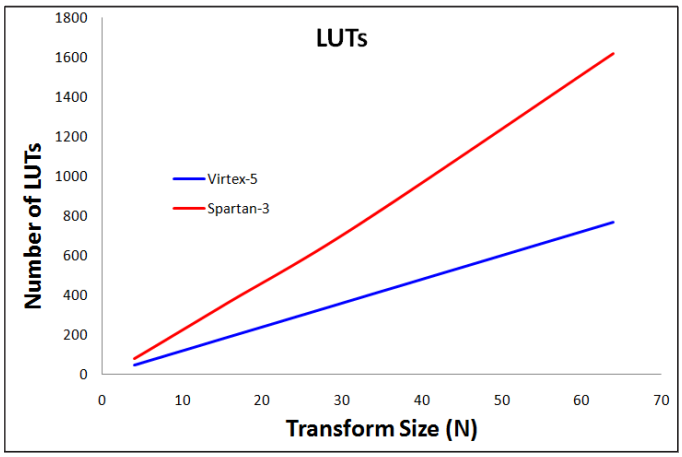


Figure 5.6: LUT utilisation with different transform sizes on different FPGA platforms

Table 5.3: FPGA implementations results comparison of the proposed design

FPGA Platforms	N	Area (Slices)	DSP48s	Flip Flop	Eq.Gtae Count	Freq. (MHz)
Virtex-4 [Proposed]	4	72	4	48	832	275.77
Virtex-4 [112]	4	48	4	96	1024	210.6

FPGA Chip Level Details

In addition to FPGA implementation, a careful manual place and route of critical nets and manual pin assignment for the designs have been performed using Xilinx PACE and Xilinx Floorplanner [2]. The FPGA chip layout of Virtex-5, Virtex-4 and Spartan-3L for proposed design is shown in Fig. 5.7 5.8 and Fig. 5.9 respectively.

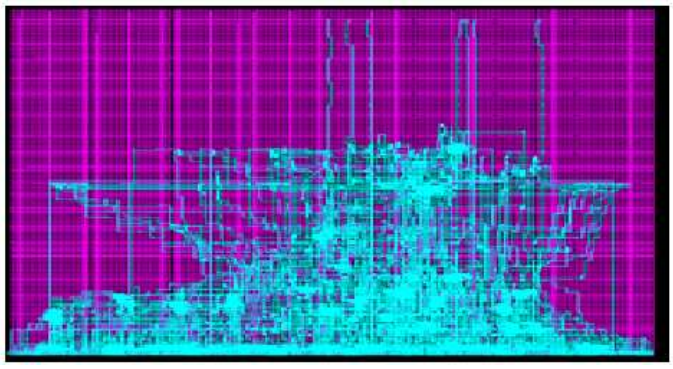


Figure 5.7: Chip layout of Virtex-5 for proposed design when the (N) is 32

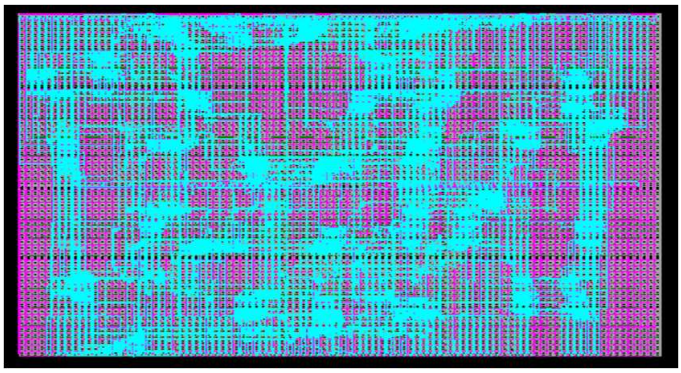


Figure 5.8: Chip layout of Virtex-E for proposed design when the (N) is 32

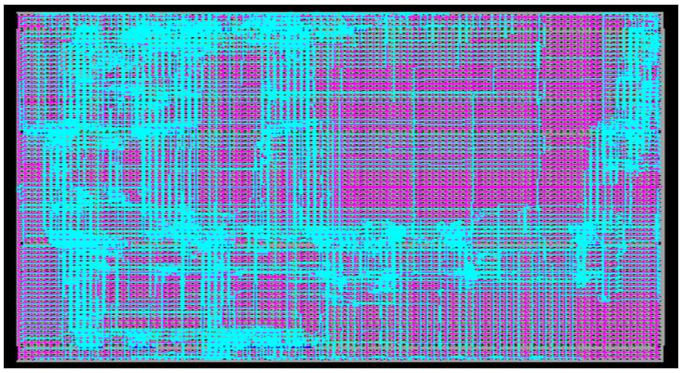


Figure 5.9: Chip layout of Spartan-3 for proposed design when the (N) is 32

5.5 Conclusion

In this Chapter, two architectures based on systolic array using parallelism and pipelining have been proposed to implement the matrix-vector products given by Eq.5.6. The first proposed architecture is a linear systolic array with pipelining process. This architecture needs ($\sum_{i=1}^{N} N_i - 1$) number of delay registers to keep the process of pipeline flow. It is not the case in the second architecture, all inputs ($q_0 q_1 ... q_{N-1}$) are fed in parallel in first $N/2$ PEs then they are shifted to next PE in every clock cycle. The only deference in second architecture is that all resources are used in first $N/2$ PEs. In this case parallelism helps to reduce the area complexity rather than the time complexity. This is because all inputs are fed in parallel and there is no need for delay registers. It worth mentioning that the second architecture reduces the number of registers by 42% compare to first architecture and outperformed other existing results in place.

The proposed pipelined architecture for cyclic convolution is implemented on different FPGA platforms with vector size (N) 4,8,16 and 32 with word-length (W=8) to see the impact of N when it is increasing. The results have shown that there is a significant differ-

ence in terms of frequency and area occupied (slices) between each FPGA platforms. For instance, Virtex5 used 40% less slices compared to Virtex-E, in other words the variation of the area in each platform depends on the technology used which can be further improved if the available resources on these platforms are used in appropriate way. The implementation results have clearly shown a significant improvement and outperformed other existing results in place. To summarise, architectural level optimisation techniques and efficient FPGA implantation for cyclic convolution and have been discussed in this chapter. In the next chapter, an in-depth evaluation of a high level power modeling approach for FPGA based designs is discussed.

Chapter 6

Functional Level Power Modelling Approach: An In-depth Evaluation

6.1 Introduction

Power budgets are becoming increasingly stringent and need higher attention in the early stages of the design cycle. The advent of battery operated devices and the increased deployment of processing in energy and thermal constrained environments such as satellites has accelerated interest in power modeling. Hardware implementations of power aware applications necessitate the design space exploration to realise an optimal solution. Field Programmable Gate Arrays (FPGAs) have been evolving and improving rapidly, and have consistently shown the fastest rates of performance gains. While capacity doubling in CPUs occurs every 18 months, in line with the prediction of the well known Moores Law, the performance of FPGAs increases at an even more rapid rate of four times every two years [2].

Design improvement is typically an iterative process that must take into account optimisation strategies at all feasible levels of abstraction. Power dissipation has become a key design issue in FPGAs based architectures because of a number of factors including: mobility, battery limitations, thermal constraints, reliability issues and cost of cooling sub-systems. In fact, power has been cited as a limiting factor in the ability of FPGAs to continue to replace Application Specific Integrated Circuits (ASICs) [113]. Todays largest

FPGAs implement complex systems with millions of gates that can consume several watts of power [114]. The most effective strategies must then be selected by analysing the impact of the different choices on a level-by-level basis, instead of just at the very end of the flow. This enables us to shorten the design flow. However, it requires the development of power modelling tools that provide reliable estimates of the power to enable the designer to make the right design choices while optimising the core. Models have always been important for electronic system design at all levels, whether at the component level (IC design), the board level or the system level.

To overcome the limitations of existing work described in Chapter 2, an in-depth power modelling evaluation based on functional level power modeling approach [115] is presented in this chapter. In this model each individual component of Dynamic Power (DP), i.e. Clock Power (CP), Signal Power (SP), Logic Power (LP), Input Power (IP) and Output Power (OP) is measured separately, and modelled individually. The power models are obtained by performing non linear regression analysis on system variables.

The rest of this Chapter is organised as follows. The underlying concepts of the proposed model is presented in Section 6.2. Brief introductions into mathematical background for presented modeling is described in Section 6.3. The presented modeling approach applied to various IP cores (HWT, FRIT and cyclic convolution) is presented in Section 1.4, 1.5 and 1.6 respectively. The proposed modelling evaluation and analysis is presented in Section 1.6. Concluding remarks are provided in Section 6.8.

6.2 Underlying Concepts of the Proposed Power Modeling

The underlying concept is to build a mathematical model that incorporates all the system variables, enabling the user to perform high level estimation of the power and energy metrics of the core for a given set of parameters early on in the design cycle itself. The steps involved in building the power model are described in Fig. 6.1. Functional level modeling is used to describe the capability of a circuit. It means the circuit is modelled as a connection of identifiable function blocks. Typical functions would be counters, decoders, memories and specific IP cores. The most significant difference between existing modeling methods and functional level approach is that our presented solution is a high level modeling technique, and is used for modeling IP cores rather than the general fab-

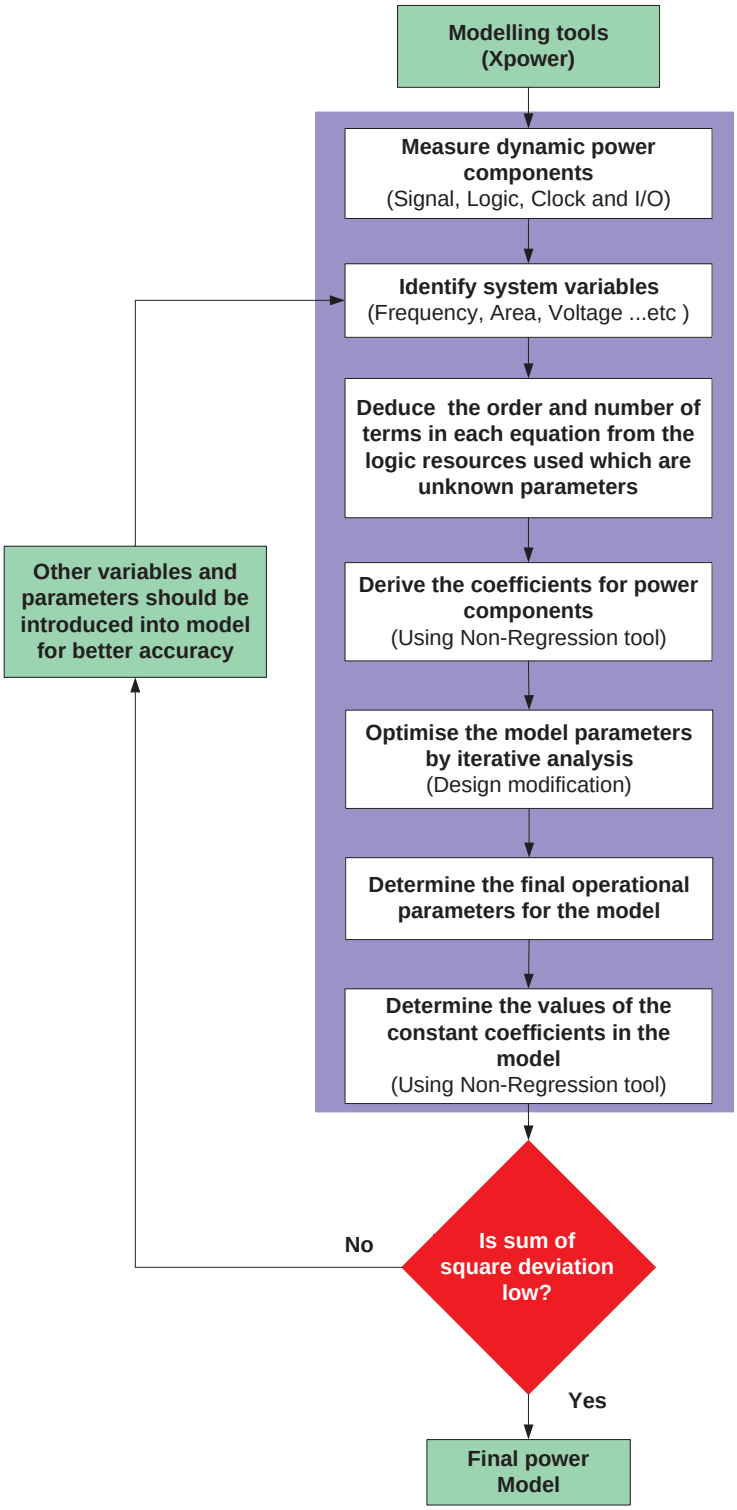


Figure 6.1: The steps involved to build the functional level modelling approach

ric of the FPGA or the ASIC. This has a number of advantages. The functional level approach methodology scales very well with changes in platforms, and even prototyping technologies. Advances in the device technology of the prototyping platform naturally

result in lower power and energy consumption metrics. Although this results in different constant coefficients for each platform, it is important to highlight that the model itself remains unchanged. The other key differentiator lies in the fact that the functional level approach tool is used by the IP core developer to optimism and model the core whereas it provides the system designer with the information that enables him to select the optimum set of core parameters to achieve the performance budgets and even to analyse if the budgets are realistic in the first place. Low level power estimation techniques, on the other hand assume that core development and core deployment are integrated, which leads the designer to perform both tasks. The functional level methodology has been successfully incorporated into a proposed design flow presented in Fig. 6.2 for obtaining power efficient implementations of FPGA based designs.

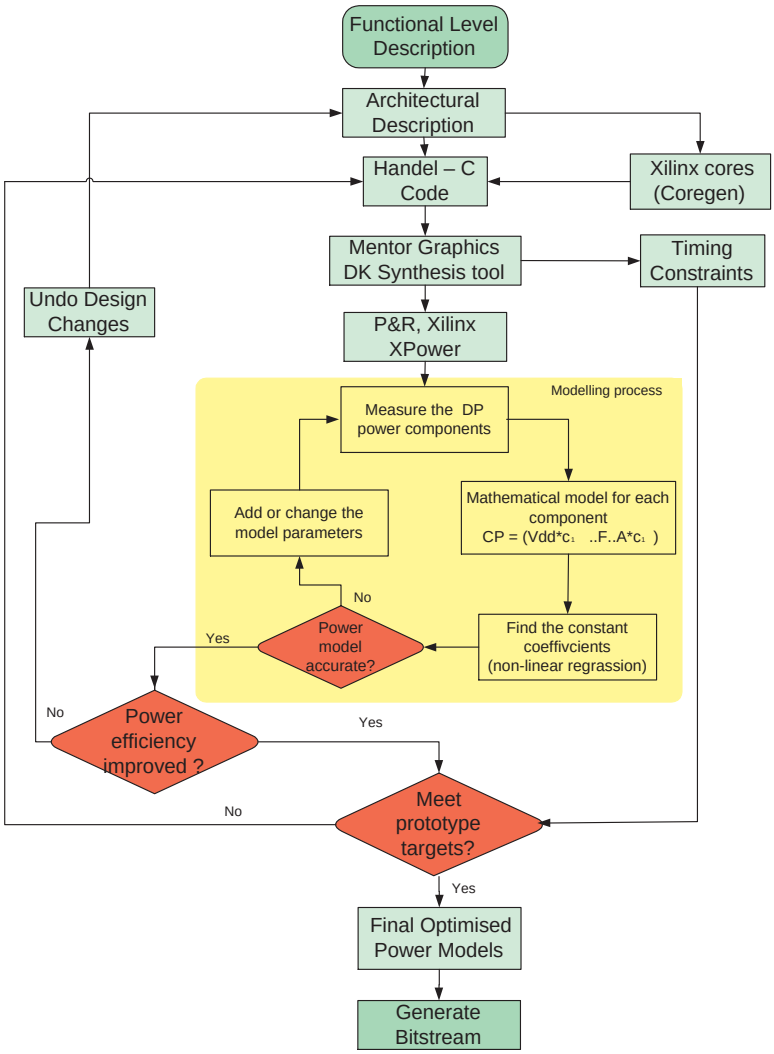


Figure 6.2: Proposed design flow for functional level approach for FPGA IP cores

Fig. 6.2 shows the over all process of functional level for proposed methodology. Based on the functional description and architectural optimization the design will be coded with hardware languages in our case Handel-C, then the synthesis tool (Mentor Graphic DK) is used to synthesize the code. Once the synthesis operation is finished the design is imported to the power modelling tools (Xpower). The modelling power tool is used to measure the estimated DP components (clock, signal, input, logic and ouput) with different frequency. On the other hand, based on the function and architecture the area will be estimated, by this time the independent variables (voltage and frequency) are also known. Having this information will leads to model mathematically each DP component with some scaling coefficient, after the mathematical model is derived the model is imported to Non-regression tool to find the value scaling coefficient and test for accuracy. If the power efficiency is not improved then the design need to improved and all the processes are repeated until power efficiency is achieved. Once the power accuracy and efficiency is satisfactory then the final model can be drawn. Based on this methodology a number of low power cores for various applications including different HWT, FRIT and cyclic convolution have been successfully evaluated using a combination proposed power model and the proposed design flow.

6.3 Mathematical Background

The mathematical background behind the evaluated methodology is derived as follows. We define the instantaneous DP dissipation of an FPGA based implementation as:

$$P_t = V_{cc} \cdot \alpha \cdot f \cdot \delta_t \cdot A \tag{6.1}$$

where V_{cc} is supply voltage, α is constant of proportionality, f is operational frequency δ_t is instantaneous activity rate and A is the area occupied.
The average power dissipated in one computational cycle ($Pc_{average}$) is then given by:

$$Pc_{average} = \frac{\int_T P_t \cdot dt}{T} \tag{6.2}$$

FPGAs DP comprises I/O, clock, logic and SP. Expanding Eq. 6.2, we get:

$$Pc_{average} = \frac{\int_T \sum_d P_{td} \cdot dt}{T} \tag{6.3}$$

where T is one complete computational cycle and d is $input, output, clock, signal, logic$ For complex design take each B separately where B is sub-block, and substituting Eq. 6.1 in Eq. 6.3, we get:

$$Pc_{average} = \frac{V_{cc} \cdot f \cdot A}{T} \sum_{b=1}^{B} \int_{T} \sum_{d} \alpha_{db} \cdot \delta_{tb} \cdot dt \tag{6.4}$$

Each component of DP is modelled individually. By separating the models and rearranging the orders of integration and summation, we get:

$$Pc_{average\_d} = \frac{V_{cc} \cdot f \cdot A}{T} \sum_{b=1}^{B} \alpha_{db} \int_{T} \delta_{tb} \cdot dt \tag{6.5}$$

Each architecture in FPGA consisting of Bs, let us consider one B which is shown in Fig. 6.3 that the signal transition probability at the output of each block at instant t to be defined by a function of present and previous inputs as follows:

$$\delta_{tb} = \phi_b(i_b(t), i_b(t-1)) \tag{6.6}$$

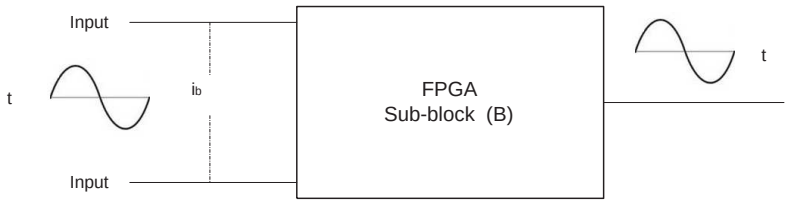


Figure 6.3: FPGA sub-block (B)

By substituting Eq.6.6 in Eq.6.5, we get:

$$Pc_{average\_d} = \frac{V_{cc} \cdot f \cdot A}{T} \sum_{b=1}^{B} \alpha_{db} \int_{T} \phi_b(i_b(t), i_b(t-1)) \cdot dt \tag{6.7}$$

Since we are interested in constructing a high level model, average power dissipation independent of signal statistics needs to be estimated. The average power dissipated over a full cycle operation is determined. If the model we construct consists of N parameters, where each parameter is represented by the symbol k, the model can be described in general as a function of k. Our model can be defined to be equivalent to the power equation as follows:

$$\Psi_d(k_1, k_2, ...k_N) = \frac{V_{cc} \cdot f \cdot A}{nT} \sum_n \sum_{b=1}^{B} \alpha_{db} \cdot \Delta_b \tag{6.8}$$

where i_b is set of inputs applied to block, Δ_b is average activity rate for each block and n: length of the maximal sequence

The estimation of each individual component of DP are obtained using Xilinx Xpower [2], and the only unknown terms in Eq.6.5 are the model coefficients and scaling coefficients α_{db}. These coefficients are determined by means of an adaptive choice of the model Hessian. The algorithm is essentially a combination of Gauss-Newton and Levenberg-Marquardt methods. The fundamental mathematical concept that is used for determining the coefficients is non-linear regression, and is implemented using NLREG [116]. This process yields an intermediate model which is used as a basis for iterative design optimisation.

6.3.1 Power Analysis

There are two primary types of power consumption in FPGAs: static and dynamic power. Static Power is consumed due to transistor leakage. Dynamic power (DP) is consumed by toggling nodes as a function of voltage, frequency, and capacitance which shown in Eq.6.9 [2]:

$$DP = \sum_i (C_i \times F_i \times V^2) \tag{6.9}$$

Where Ci and Fi are the capacitance and average toggle rate of the ith net, and V is the internal voltage. The DP for different FPGA platform is presented in Fig. 6.4 and the power DP components is presented in Fig. 6.5 respectively. It ia worth noting that the power is measured by using Xilinx XPower [2]. It can be seen from Fig.6.4 that the DP dissipation for each platform is directly proportional to the frequency used. The higher frequency used the more power dissipated. It also verifies that the power consumption has been minimised with the most recent and advanced FPGA platforms. Virtex5 consumed 15% less power than Virtex4 and almost 50% less than Virtex-E (V2000E). It worth mentioning that DP shown in Fig.6.4 is measured based on four mean components (clock, signal, logic and input) which meanly dominate the total dynamic power.

Fig.6.5 illustrates the estimated power consumption for different components of DP on V2000E platform. From Fig.6.5 it can be seen that signal power and logic power are the

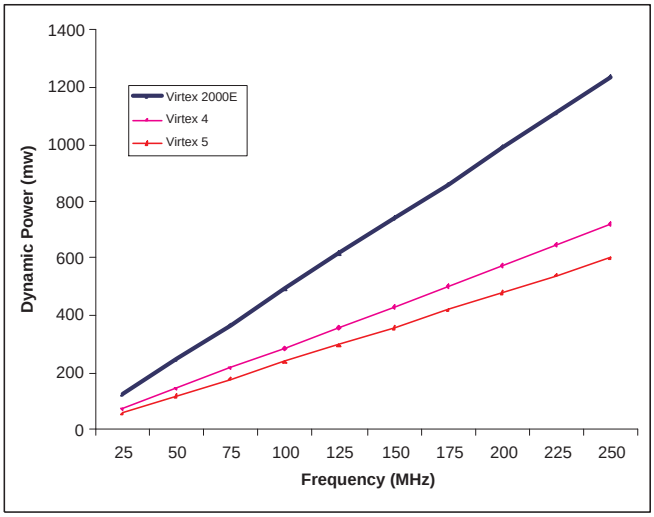


Figure 6.4: Power diagram for different FPGA platforms

mean two components which dominating the DP. Based on this results 85% of the DP is formed by signal and logic powers. It also shows that all the DP components are directly proportional to frequency.

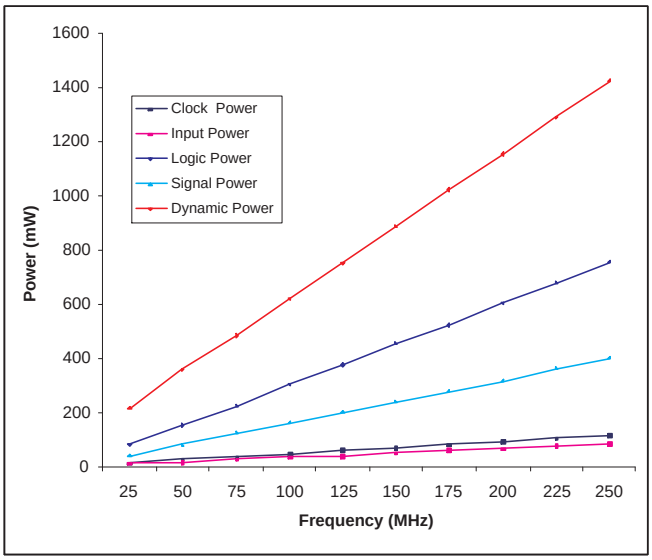


Figure 6.5: Power diagram for different components of DP on Virtex-2000E

6.4 Proposed Modeling Methodology Applied on HWT Core

In the section the power modeling for HWT core based on DA is presented. The architectural and implementation details have been presented in Chapter 3. The only further opportunity for optimisation that has been exploited is the use manual PAR of critical nets and manual pin assignment for the designs has been performed using Xilinx Floorplanner [2]. This process yields compact and optimised design with short nets, and serves two important purposes. Firstly, short nets have less propagation delay, and upto 25% gains in maximum frequency have been achieved. Second, short nets have less parasitic capacitance and DC load, and therefore dissipate less power than long nets. Manual pin assignment also enables us to locate the I/O pads close to the design area.

Power modelling is performed for implementations on the Virtex-5 (XUPV5-LX110T) and Spartan-3 (3S1500L-4) FPGA chips. Differences in implementation results obtained are due to different chip topologies (area, pin distribution, etc). However, since the design remains unaltered, the same model can be used to define the power consumption for all implementations. The corresponding models are derived by performing non-linear regression analysis on the power data obtained. On iterating the regression until convergence is achieved, the values of the scaling coefficients in the models can be determined. By back substitution of these values back into the model, a global equation that defines the power consumption of the system for any given combination of system parameters has been derived. The power measurements data on which the models are based are graphically represented in the following subsections.

6.4.1 HWT Area Modeling Details

The area occupied by the implementation of HWT is represented in terms of slices, which depends on two components in the architecture: the control logic area and the ROM area. The ROM area increases exponentially with vector length. It is also proportional to logarithm of vector length. This term is represented by the variables associated with coefficient $\alpha 1$ in Eq. 6.10. The area occupied by the control logic and arithmetic blocks (shifters, adders etc.) is proportional to the vector length, and is represented by the variables associated with coefficient $\alpha 2$. Coefficient $\alpha 3$ is introduced to accommodate unrepresented additional areas of the architectures to balance the models effectively. Based

on these observations, the area model for the proposed HWT architecture is described in Eq. 6.10:

$$TA = \alpha 1 \cdot N + \alpha 2 \cdot (\log_2(255 * N)) + \alpha 3 \tag{6.10}$$

where TA is the total area occupied by the proposed architecture, $\alpha 1, \alpha 2$ and $\alpha 3$ are the scaling coefficients in the area model.

6.4.2 Clock Power Model

Clock Power (CP) in FPGAs depends on the distribution of the clock nets (which depends on the chip area over which the design is spread out). Other factors influencing clock power are chip frequency f, and voltage v. Based on these parameters, the same clock power model holds true for FPGA platforms. From the power data pictorially represented in Fig. 6.6, an appropriate model has been selected and is described in Eq. 6.11. Where $c1, c2$ and $c3$ are scaling coefficients in the clock power model.

$$CP = c1 \cdot v^2 \cdot TA + c3 \cdot f + c2 \cdot f + c3 \tag{6.11}$$

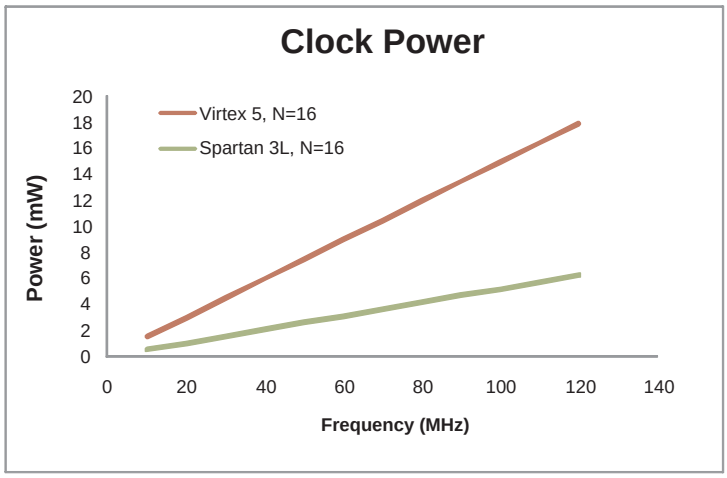


Figure 6.6: Clock power chart for 2D HWT when the transform (N) size is 16 at different frequencies

6.4.3 Signal Power Model

Signal Power (SP) is proportional to the number and length of nets over which signal switching occurs. It also depends on the voltage levels between which the switching occurs, as well as the frequency of switching. Taking into consideration these parameters, and the SP data presented in Fig. 6.7, the SP model for the proposed HWT core is defined in Eq. 6.12 where $s1, s2, s3$ and $s4$ are scaling coefficients.

$$SP = s1 \cdot v^2 \cdot TA^{s2} \cdot f \cdot s3 \cdot f + s4 \tag{6.12}$$

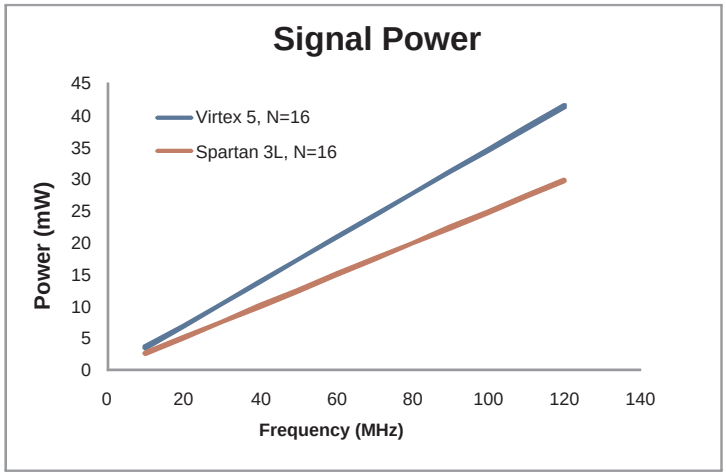


Figure 6.7: Sinal power chart for 2D HWT when the transform (N) size is 16 at different frequencies

6.4.4 Logic Power Model

Logic Power (LP) consumption is a function of the number of slices occupied, chip frequency and voltage. By observing and interpreting the power data presented in Fig. 6.8, the best model describing the LP for the proposed HWT architecture is described in Eq. 6.13. Where $l1, l2, l3$ and $l4$ are scaling coefficients in the LP model.

$$LP = l1 \cdot v^2 \cdot TA + f + l2 \cdot f + l3 \tag{6.13}$$

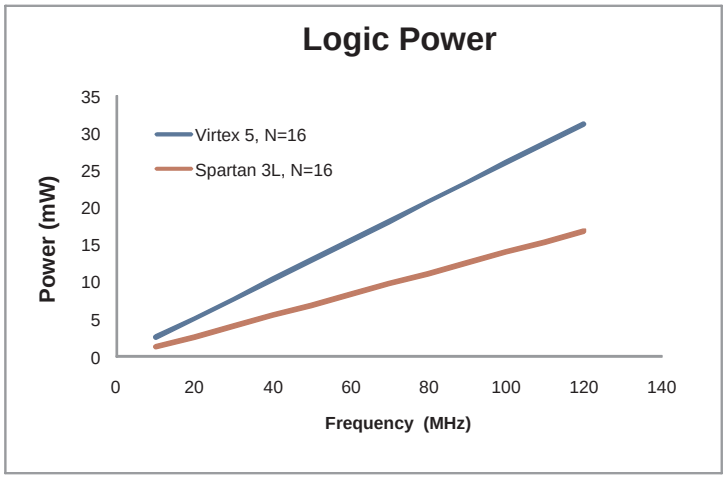


Figure 6.8: Logic power chart for 2D HWT when the transform (N) size is 16 at different frequencies

6.4.5 Input Power Model

Input Power (IP) depends on the number of input pins in the design, the block size, chip voltage, and input frequency. The corresponding model that satisfies the curve fitting requirements of the HWT architecture for the data presented in Fig. 6.9 is defined in Eq. 6.14, where $i1, i2$ and $i3$ are scaling coefficients. It worth mentioning that the input buffers of the Spartan 3L platform dissipate no power, and hence the coefficients in the corresponding models for this platform are zero.

$$IP = i1 \cdot v^2 \cdot f + i4 \cdot (N)^{i2} + i3 \tag{6.14}$$

6.5 Proposed Modeling Methodology Applied on FRIT Core

Modelling of FRIT with parallel FRAT core whose architectural and implementation details have been explained in detail in Chapter 4 is presented in this section. The corresponding parameters that influence the choice of the model have been explained in the following subsections.

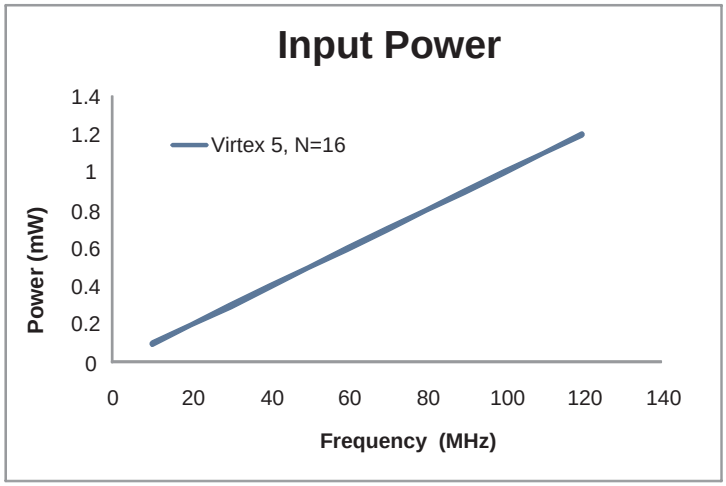


Figure 6.9: Input power chart for 2D HWT when the transform (N) size is 16 at different frequencies

6.5.1 Area Model for FRIT

The area occupied by the design in term of number of slices depends on two components in the architecture: the control logic area and the memory area where the vector values are stored. The area occupied by memory increases exponentially with vector length. It is also proportional to logarithm of vector length. This term is represented by the variables associated with coefficient $\alpha 1$ in Eq. 6.15. The area occupied by the control logic and arithmetic blocks (shifters, adders etc.) is proportional to the vector length, and is represented by the variables associated with coefficient $\alpha 2$. Coefficient $\alpha 3$ is introduced to accommodate unrepresented additional areas of the architectures; and to balance the models effectively. Based on these observations, the area model for the proposed FRIT architecture is described in Eq. 6.15:

$$TA = \alpha 1 \cdot p(p+1) \cdot \lceil (\log_2(255 * p)) \rceil + \alpha 2 \cdot \lceil (\log_2(255 * p)) \rceil + \alpha 3 \qquad (6.15)$$

where TA is the total area occupied by the proposed architecture, $\alpha 1, \alpha 2$ and $\alpha 3$ are the scaling coefficients in the area model. It has been observed that there are minor variations in the number of occupied slices for different FPGA platforms. This is because of differences in the number of slices per CLB in different platforms. Correspondingly, the logic capacity of each CLB and interconnect structure between CLBs also differ. Placement and routing takes into account these variations in the FPGA fabric and results in differences in the area metrics for the same architecture on different platforms.

It is interesting to note that the very same area model holds true for the FPGA platforms. The model equations are identical, and achieve 99% accuracy for all three platforms. The differences in the fabric of these different platforms are accounted for by the coefficients in the model, which naturally vary across different platforms. The modeling methodology is high level as it is not influenced by low level details such as routing and circuit fabric.

6.5.2 Clock Power Model

Clock Power (CP) in FPGAs depends on the distribution of the clock nets (which depends on the chip area over which the design is spread out). Other factors influencing clock power

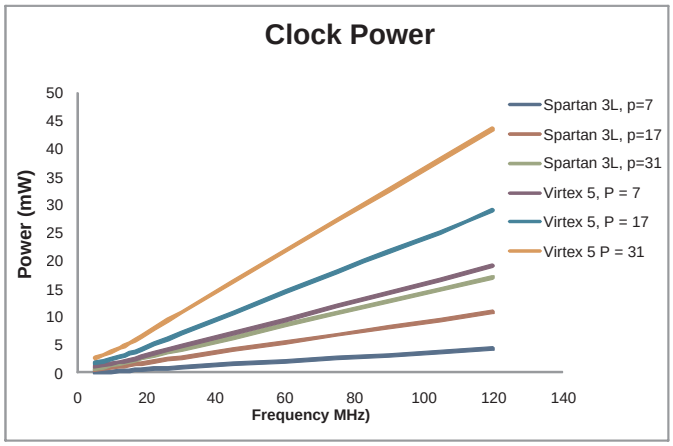


Figure 6.10: Clock power chart for FRIT when the block (P) size is 7, 17 and 31 at different frequencies

are chip frequency f, and voltage v. Based on these parameters, the same clock power model holds true for FPGA platforms. From the power data pictorially represented in Fig. 6.10, an appropriate model has been selected and is described in Eq. 6.16 where $c1, c2$ and $c3$ are scaling coefficients in the clock power model.

$$CP = c1 \cdot v^2 \cdot TA \cdot f + c2 \cdot f + c3 \tag{6.16}$$

6.5.3 Signal Power Model

Signal Power (SP) is proportional to the number and length of nets over which signal switching occurs. It also depends on the voltage levels between which the switching occurs, as well as the frequency of switching. Taking into consideration these parameters, and the

SP data presented in Fig. 6.11, the SP model for the proposed FRIT core is defined in Eq. 6.17 where $s1, s2, s3$ and $s4$ are scaling coefficients.

$$SP = s1 \cdot v^2 \cdot TA^{s2} \cdot f + s3 \cdot f + s4 \tag{6.17}$$

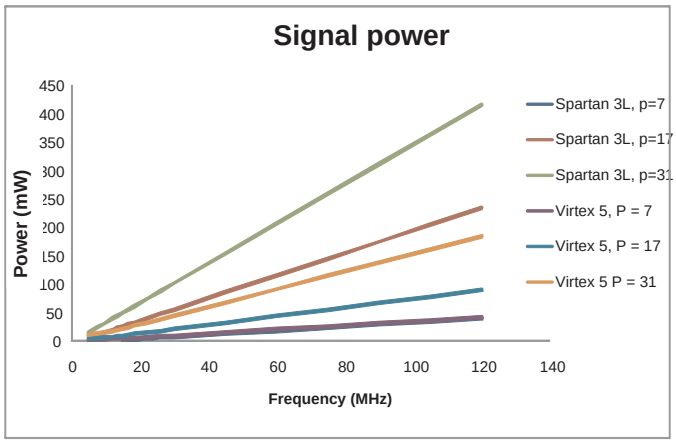


Figure 6.11: Signal power chart for FRIT when the block (P) size is 7, 17 and 31 at different frequencies

6.5.4 Logic Power Model

Logic Power (LP) consumption is a function of the number of slices occupied, chip frequency and voltage. By observing and interpreting the power data presented in Fig. 6.12,

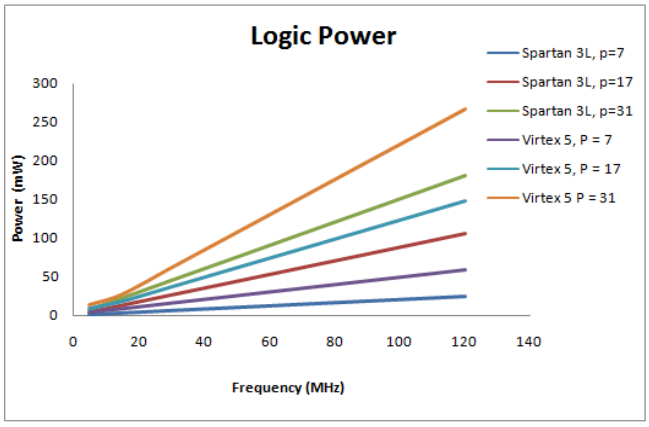


Figure 6.12: Logic power chart for FRIT when the block (P) size is 7, 17 and 31 at different frequencies

the best model describing the LP for the proposed FRIT architecture is described in Eq. 6.18 where $l1, l2, l3$ and $l4$ are scaling coefficients in the LP model.

$$LP = l1 \cdot v^2 \cdot TA_P \cdot f + l2 \cdot f + l3 \tag{6.18}$$

6.5.5 Input Power Model

Input Power (IP) depends on the number of input pins in the design, the block size, chip voltage, and input frequency. The corresponding model that satisfies the curve fitting requirements of the FRIT architecture for the data presented in Fig. 6.13 is defined in Eq. 6.19 where $i1, i2$ and $i3$ are scaling coefficients. It worth mentioning that by changing the block size P the number inputs do not change the power for $P = 7$, $p = 17$ and $p = 31$ are identical.

$$IP = i1 \cdot v^2 \cdot f \cdot (p^2)^{i2} + i3 \tag{6.19}$$

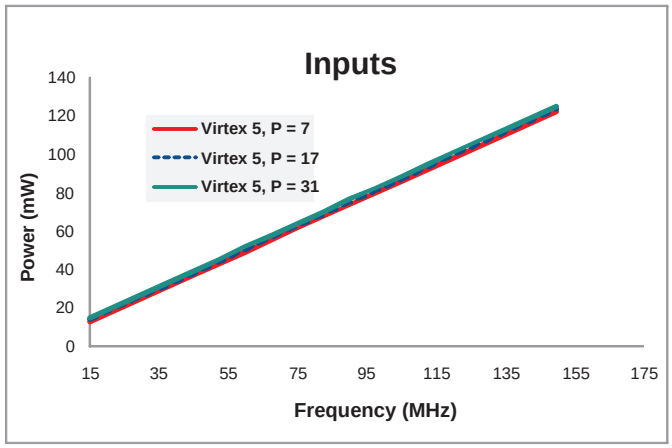


Figure 6.13: Input power chart for FRIT when the block (P) size is 7, 17 and 31 at different frequencies

6.5.6 Output Power

The Output Power (OP) depends on the number of output pins in the design, the output block size, output voltage V_{cco}, and frequency f. The corresponding model for OP is described in Eq. 6.20 where p is the block size, of the output vector, where $o1, o2$ and $o3$ are scaling coefficients in the output power model.

$$OP = o1 \cdot (v^2 \cdot N) \cdot f^2 \cdot \lceil (log_2(255 \cdot p)) \rceil \cdot p \cdot (p+1)(o2/2) + o3 \tag{6.20}$$

6.6 Proposed Modelling Methodology Applied on Cyclic Convolution Core

Modelling of cyclic convolution core whose architectural and implementation details have been explained in detail in Chapter 5 is presented in this section. The corresponding parameters that influence the choice of the model have been explained in the following subsections.

6.6.1 Area Model for Cyclic Convolution

The area occupied by the design in term of number of slices depends on two components in the architecture: the control logic area and PEs. Since it is linear architecture the area occupied is proportional to vector length. It means the area occupied by PEs is proportional to the vector length, and is represented by the variables associated with coefficient $\alpha 1$ and $\alpha 2$ in Eq. 6.21. Coefficient $\alpha 3$ is introduced to accommodate unrepresented additional areas of the architectures. Based on these observations, the area model for the proposed cyclic convolution architecture is generated based on the contained of the PEs (registers, multipliers and adders) rather than the number of PEs. The proposed area model for cyclic convolution is described in Eq. 6.21:

$$TA = \alpha_1.(N)W + \alpha_2((2N-1) + (\sum_{i=1}^{N} n_i - 1))W + \alpha_3 \tag{6.21}$$

where TA is the total area occupied by the proposed architecture, $\alpha 1, \alpha 2$ and $\alpha 3$ are the scaling coefficients in the area model, N is the vector size and W is the word length. It has been observed that there are minor variations in the number of occupied slices for different FPGA platforms. This is because of differences in the number of slices per CLB in different platforms. Correspondingly, the logic capacity of each CLB and interconnect structure between CLBs also differ. Placement and routing takes into account these variations in the FPGA fabric and results in differences in the area metrics for the same architecture on different platforms.

6.6.2 Clock Power Model

Clock Power (CP) in FPGAs depends on the distribution of the clock nets (which depends on the chip area over which the design is spread out). Other factors influencing clock power

are chip frequency f, and voltage v. Based on these parameters, the same clock power model holds true for FPGA platforms. From the power data pictorially represented in Fig. 6.14, an appropriate model has been selected and is described in Eq. 6.22 where $c1, c2$ and $c3$ are scaling coefficients in the clock power model.

$$CP = c1 \cdot v^2 \cdot TA \cdot \ f + (f \cdot c2) \cdot f + c3 \tag{6.22}$$

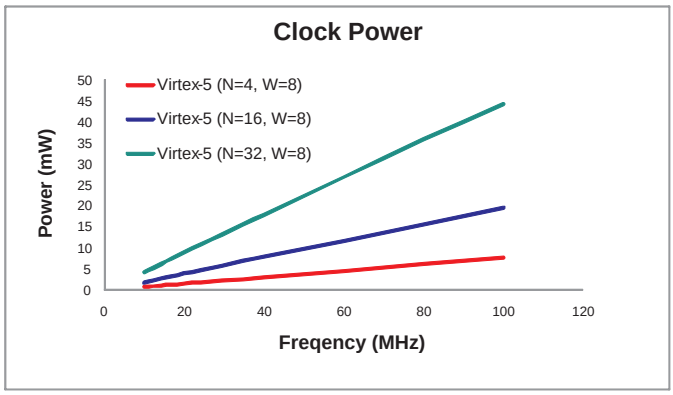


Figure 6.14: Clock power chart for cyclic convolution when the transform size (N) is 4, 16 and 32 at different frequencies.

6.6.3 Signal Power Model

Signal Power (SP) is proportional to the number and length of nets over which signal switching occurs. It also depends on the voltage levels between which the switching occurs, as well as the frequency of switching. Taking into consideration these parameters, and the SP data presented in Fig. 6.15, the SP model for the proposed cyclic convolution core is defined in Eq. 6.23 where $s1, s2, s3$ and $s4$ are scaling coefficients.

$$SP = s1 \cdot v^2 \cdot TA^{s3} \cdot f + s2 \cdot f + s3 \tag{6.23}$$

6.6.4 Logic Power Model

Logic Power (LP) consumption is a function of the number of slices occupied, chip frequency and voltage. By observing and interpreting the power data presented in Fig. 6.16, the best model describing the LP for the proposed cyclic convolution architecture is described in Eq. 6.24. $l1, l2, l3$ and $l4$ are scaling coefficients in the LP model.

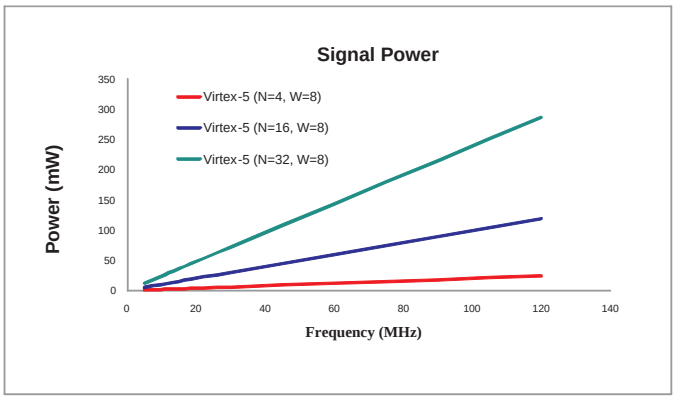


Figure 6.15: Signal power chart for cyclic convolution when the transform size (N) is 4, 16 and 32 at different frequencies.

$$LP = l1 \cdot v^2 \cdot TA \cdot f + l2 \cdot f + l3 \tag{6.24}$$

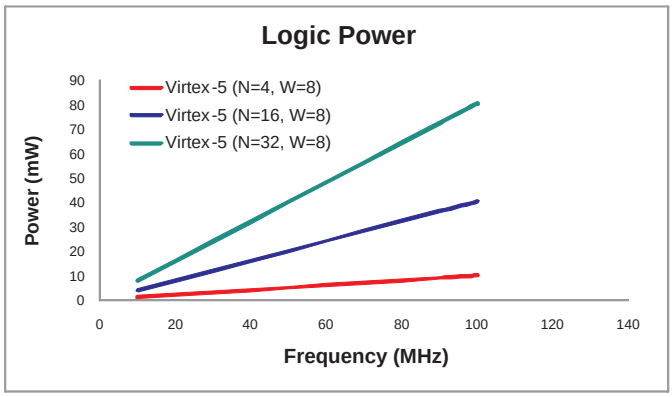


Figure 6.16: Logic power chart for cyclic convolution when the transform size (N) is 4, 16 and 32 at different frequencies.

6.6.5 Input Power Model

Input Power (IP) depends on the number of input pins in the design, the block size, chip voltage, and input frequency. It is worth noting that by increasing the input the pins are changing but the input bits are constant (8-bits). The corresponding model that satisfies the curve fitting requirements of the cyclic convolution architecture for the data presented in Fig. 6.17 is defined in Eq. 6.25 where $i1, i2$ and $i3$ are scaling coefficients.

$$IP = i1 \cdot v^2 \cdot f \cdot (2N)W \cdot i2 + i3 \tag{6.25}$$

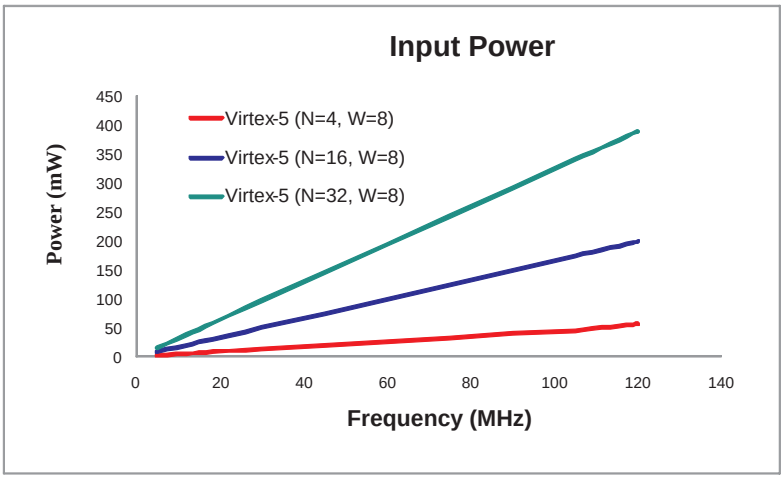


Figure 6.17: Input power chart for cyclic convolution when the transform size (N) is 4, 16 and 32 at different frequencies.

6.7 Modeling Evaluation and Analysis

In this section, the characteristics functional level modeling approach parameters are evaluated. The results from non linear regression have been analysed. In addition, the presented model has been compared with other approach in terms of accuracy.

6.7.1 Functional Level Approach Model Accuracy

It is interesting to note that the models are platform independent, and differences in the chip fabric are accounted by the model coefficients. The proportion of variance of the models (R^2) is presented in Table 6.1. (R^2) is the fraction of variance, the factors affect are deflation, logging, seasonal, adjustment and differencing. It is interesting to note that

Table 6.1: Functional level approach model accuracy for the optimised FRIT and HWT IP cores implemented on different platforms

	Spartan-3L		**Virtex-5**	
	FRIT	**HWT**	**FRIT**	**HWT**
Clock	99.89%	98.12%	99.89%	97.91%
Logic	99.10%	99.99%	99.75%	99.96%
I/P	99.60%	98.23%	98.92%	99.99%
O/P	99.72%	99.88%	99.88%	99.86%
Signal	99.45%	99.90%	99.89%	99.45%
Average	99.55%	99.22%	99.67%	99.43%

the accuracy of FRIT core is higher than the HWT core, it is because FRIT used more logic than HWT. In other word the model will be more accurate if the core utilise large

amount of resources. Additionally, it is important to highlight that the accuracy of the

Table 6.2: Functional level approach model accuracy comparison with Xilinx Xpower for the optimised FRIT and HWT IP cores on different FPGA platforms

	Spartan-3L		Virtex-5	
	FRIT	**HWT**	**FRIT**	**HWT**
Clock	2.00%	0.80%	2.20%	1.10%
Logic	1.30%	1.00%	1.80%	1.20%
I/P	1.40%	0.70%	2.40%	1.00%
O/P	2.00%	1.30%	3.20%	1.20%
Signal	2.20%	1.60%	2.60%	1.40%
Average	1.78%	1.08%	2.44%	1.18%

power models derived are relative to the accuracy of the power measurements / estimates obtained, and do not indicate the absolute accuracy by themselves. In the case of models built on Xilinx Xpower [2] estimation data, we must take into account the accuracy of the XPower estimate as well, while calculating the absolute accuracy of the models. For models based on chip power measurements, the accuracy of the models is limited to the resolution of the power supply and measuring tools used. A functional level approach model accuracy comparison with Xilinx Xpower [2] for the optimised FRIT and HWT IP cores on different FPGA platforms is shown in Table 6.2. The non linear regression modeling observation for different FPGA platforms is shown in Table 6.3.

Table 6.3: Regression modeling observation of FRIT with different FPGA platforms

	No. Observation	**FPGA Platforms.**	**Squared Deviation**	**Error Estimated**	**Average Deviation**
Clock	18	Spartan-3L	2.27E-001	1.23E-01	9.60E-2
Logic	18	Spartan-3L	1.13E+001	8.68E-01	6.71E-1
Signal	18	Spartan-3L	8.39E+002	7.47E+0	5.45E+0
Input	18	Spartan-3L	n/a	8.68E-01	6.71E-01
Output	18	Spartan-3L	2.12E+001	2.48E-01	5.71E-01
Clock	18	Virtex-5	4.27E-001	2.23E-02	2.60E-2
Logic	18	Virtex-5	1.13E+01	4.68E+01	2.71E+1
Signal	18	Virtex-5	8.39E+002	7.47E+0	5.45E+0
Input	18	Virtex-5	1.43E+02	2.48E-01	5.71E-01
Output	18	Virtex-5	4.22E+01	1.48E-01	4.11E-01

The scaling coefficient values of the power models are obtained by performing non linear regression until convergence, and are presented in Table 6.4 and 6.5 respectively.

Table 6.4: Values for scaling coefficient of power models for the proposed FRIT IP core architecture on FPGA Platforms

	Spartan-3L	Virtex-5
c1	4.39E-005	1.396E+00
c2	1.649E-02	-3.331E+03
c3	-9.33E-02	3.000E-03
s1	2.393E-04	3.087E-01
s2	1.143E+00	-2.990E+02
s3	1.143E+00	-8.999E-02
s4	8.498E-01	-3.441E-01
l1	1.607E-03	7.030E-03
l2	5.082E-02	5.095E+02
l3	3.810E-01	5.203E-01
i3	2.113E+01	1.991E-02
i4	3.448E-001	2.421E-01
i1	3.617E-03	4.130E-02
o3	3.14E+001	-3.949E-01
o4	4.498E-01	-2.11E-01
o1	2.227E-03	3.230E-02

Table 6.5: Values for scaling coefficient of power models for the proposed HWT IP core architecture on FPGA Platforms

	Spartan-3L	Virtex-5
c1	3.290E-005	2.126E+00
c2	1.449E-02	-3.452E+03
c3	-7.13E-02	2.900E-03
s1	2.393E-04	2.687E-01
s2	1.243E+00	-2.591E+02
s3	1.343E+00	-8.399E-02
s4	6.898E-01	-3.241E-01
l1	2.007E-03	6.030E-03
l2	4.992E-02	4.995E+02
l3	4.010E-01	4.903E-01
i3	0.002E+01	2.191E-02
i4	0.001E-001	2.621E-01
i1	0.002E-03	3.930E-02

6.7.2 Comparison Modeling Characteristics of Functional Level Approach with Existing Work

Unlike most high power modeling the functional level approach modeling methodology scales very well with changes in platforms, and even prototyping technologies. Advances in the device technology of the prototyping platform naturally result in lower power and

energy consumption metrics. Although this results in different constant coefficients for each platform, it is important to highlight that the model itself remains unchanged. A comparison of functional level approach with existing FPGA power modeling methodologies is presented in Table 6.6.

Table 6.6: Comparison modeling characteristics of different approaches with functional level approach model

	Avg r^2	Level	Scaleable	Platform Agnostic
Proposed	.99	High	Yes	Yes
[25]	.97	High	No	Yes
[24]	Low	Arch.	Yes	No
[117]	.95	High	No	No
[118]	.95	RTL	No	No
[27]	.82	High	Yes	No
[82]	Various	High	No	Yes
[28]	.97	High	No	No
[29]	High	Arch.	Yes	Yes
[31]	.92	Mixed	Yes	No

6.7.3 Observations and Analysis

From the power model graphs, it can be seen that the power is directly proportional to frequency f for clock, signal, logic and input. In addition, it can be also clearly seen that the models are robust and can consistently provide good estimates of power dissipation within the design space for all platforms. This clearly shows the scalability of the proposed power models. The differences in the FPGA fabric are low level / circuit level characteristics, and should have no effect on the models since the proposed model is a high level modelling methodology. The architectural differences, and consequent variations in power consumption across different platforms are accounted by the coefficients in the models as shown in Tables 6.4 and 6.5 respectively. Table 6.1 presents the model accuracy for all power components, it can been seen that the model accuracy is almost 99% true for all power components. The results for proposed model are also compared with Xilinx Xpower tools, it can be seen from Table 6.2 that the margin of difference between Xpower and proposed is very small.

6.8 Conclusion

In this Chapter, an in-depth evaluation and application of a high level IP core macromodelling methodology called functional level power modeling approach that overcomes some of the weakness in other existing modelling methodologies for FPGA based designs have been presented. The mathematical techniques that form the basis of the proposed power modeling has been validated by a range of custom IP cores which have been developed in previous Chapters. Unlike most high level power modeling the functional level modeling methodology scales very well with changes in platforms, and even prototyping technologies. The differences in the FPGA fabric are low level/circuit level characteristics, and should have no effect on the models since the proposed model is a high level modelling methodology. It has been observed that the power is directly proportional to frequency f for clock, signal, logic and input. Based on the results achieved, the proposed model accuracy is almost 99% true for all DP components. The results for proposed model have also been compared with Xilinx Xpower tools and the difference between Xpower and proposed were very small. A characteristics comparison of the proposed model has also been carried out with other existing models in place. In the next chapter, concluding remarks and future suggestions about the work presented in this thesis will be presented.

Chapter 7

Conclusions and Future Work

7.1 Introduction

Advanced image and signal processing techniques in a wide range of disciplines and applications, from computer vision and medical imaging to image and video compression, are replacing previous generations' technology offering enhancements, such as better streaming capability, higher compression for a given quality and lower latency. Many of these operations such as transformations of images and signal analysis, classification, communication, etc are matrix transforms, which occur frequently in a wide variety of real world algorithms used in digital image and video processing applications [77, 119].

Researchers are working round the clock to develop efficient algorithms and architectures suitable for these applications. Therefore, application designers face many new and difficult challenges as they attempt to deploy technology that can execute high-performance computations, manipulate larger and larger data sets and visualise the complex data in a better way. Multiresolution algorithms are among these algorithms to be carefully considered. Multiresolution algorithms are highly suitable for a number of image processing applications as described in previous Chapters. They offer efficient implementation for these applications but the processing time still remain peculiar. Researchers are working round the clock to fine the best solution for these algorithms. Recently, Field Programable Gate Arrays (FPGAs) are gaining the momentum to be used for acceleration of these applications because of the parallelism offered by these devices which can perform mathematical operations on an entire vector or matrix at the same time [120].

Among multiresolution algorithms ridgelet and curvelet transforms [46–48] have been gen-

erating a lot of interest due to their superior performance over wavelets. Wavelets have been very successful in applications such as denoising and compact approximations of images but they do not isolate the smoothness along edges that occurs in images. These shortcomings of wavelets are well addressed by the ridgelet and curvelet transforms, as they extend the functionality of wavelets to higher dimensional singularities, and are effective tools to perform sparse directional analysis.

However, it is worth mentioning that the nature of the applications and algorithms that have been targeted in this work very often impose serious limitations on the amount of hardware involved and the rate of power consumption due to some factors including mobility constraints, cost performance trade-offs, and etc. Thus, there has been a continued effort to meet the conflicting challenges of ever-growing computational demand with minimal utilisation of hardware resources and power [121].

The main goal of the work reported in this thesis is to exploit techniques at the algorithmic and architectural level to deliver highly optimised and efficient multiresolution algorithms for FPGA based designs, suitable for use in both general purpose and specific image (medical) and signal processing problems. Another key objective is to address the issues of dynamic power dissipation in FPGAs through the evaluation and application of a high level power modelling approach technique encapsulated within a suitable design flow to optimise and model the aforementioned multiresolution algorithm cores. The proposed architectures have been implemented using hybrid approaches, for instance in most of the design Handel-C, Xilinx CoreGen cores (adders and multipliers) are combined in order to take advantage of each design entry and then prototyped on the RC1000 and RC10 FPGA boards and implemented on Virtex-4 and Virtex-5 without any architecture modification.

The rest of this chapter is arranged as follows. The evaluation of results and contributions obtained throughout this research are summarised and evaluated in Section 7.2. Some possible routes to be investigated for a future extension of this work are also provided in Section 7.3.

7.2 Evaluation of Results and Contributions

The preceding Chapters described different design methodologies used for the efficient design and implementation of various transform methods and signal processing algorithms on FPGAs. A brief analysis of FPGAs power and an in-depth evaluation of presented high

level power modelling for FPGAs have also been discussed. This section is concerned with the evaluation of the work presented in these Chapters.

7.2.1 Measurement of Success

In this project the measurement and comparison of new proposed architectures with existing implementations are presented. The comparison was based on the computation time, area required, throughput rate and power dissipation; all of which depends on the the various design optimisation strategies pursued and design parameters such as word-length of the input data and transform size. In the case of the implementation of these architectures, the comparison was based on the number of CLBs, slices, LUTs, the maximum running frequency of the design and the power. The proposed architectures were implemented and synthesised on different FPGA devices in order to make a fair and consistent comparison with existing cores using the same platform. A functional level power modeling approach has been developed in response to the lack of availability of a similar tool for FPGA based designs, which precludes the possibility of a direct comparison with existing methodologies. However, an objective comparison based on model accuracy and a number of other parameters such as modelling level, scalability and platform independence has been performed to evaluate the ability of the presented model to perform stated objectives.

7.2.2 Results Achieved

In Chapter 2, a set of goals were specified which would determine the success of the work presented in this thesis, namely:

- Novel architectures for multiresolution analysis algorithms using advanced arithmetic techniques and design methodologies through the optimisation strategies at various abstraction levels have been developed;
- Scalable, parameterisable, efficient FPGA based multiresolution IP cores suitable for use in both general and medical imaging applications have been designed and implemented;
- Optimisation strategies at various abstraction levels to analyse the effectiveness of these techniques for performance enhancement and power reduction have been

investigate and applied;

- The best performance trade-offs such as area/speed for the FPGA implementations of these presented cores have been investigated; and
- An in-depth evaluation of a high level power modelling in terms accuracy for proposed multiresolution algorithms IP cores has been presented.
- In Chapter 3, high performance and power efficient architectures of Haar Wavelet Transform (HWT) based on factorisation methodologies using Distributed Arithmetic (DA) principles, which is suitable for FPGA implementation [95, 96, 122] has been presented. The proposed architectures have been implemented on different FPGA platforms for evaluation. In order to evaluate, the designs have been also implemented on low power and advanced FPGAs. The evaluation of the implementation results has shown that the obtained results have outperformed existing implementations in place.
- In Chapter 4, architectural techniques such as parallelism, and pipelining have been exploited to yield efficient and power aware architectures for Finite Radon Transform (FRAT) and Finite Ridgelet Transform (FRIT) [123]. The proposed architectures have been implemented using different FPGA platforms. The performances have been evaluated with other existing work in place. The architectural and FPGA implementation results have outperformed some other existing in various key performance metrics.
- In Chapter 5, architecture techniques such systolisation, parallelism and pipelining have been explored to yield a power aware and optimised architectures for cyclic convolution. Efficient FPGA implementations of the proposed pipelined architecture for cyclic convolution have been also presented, the architectural and FPGA implementation results are compared with other existing work in place. The results clearly showsn a significant improvement and outperformed other existing results in place.
- In Chapter 6, an in-depth evaluation of high level IP core macromodelling methodology called functional level power modeling approach [123] has been presented. The model evaluated in this chapter has overcome some of the weakness in other existing FPGA based modelling methodologies. The model has successfully verified on the

various FPGA IP Cores presented in preceding chapters (HWT, FRIT and cyclic convolution).

It can therefore be claimed that the project has made significant progress in meeting the key stated objectives.

7.2.3 Limitations

The objectives stated in Chapter 2 have been met and fully achieved. In the meantime, a few of restrictions and limitations have been raised during the development of this research project:

- With some more effort and time the architectures developed for multiresolution algorithms can be modified further in terms of efficiency and power dissipation. For instance, since the HWT coefficients are mtrix based it can be implemented using systolic array architecture or using parallelism to access more than one ROM at the time which helps to seep up the process. For FRIT and HWT it would have been very nice to see the impact of cyclic convolution as main building block. The cyclic convolution can be improved further by introducing a multiplexer to select the inputs rather than parallel inputs which use all the inputs of FPGA pins with a small design.

- Real hardware implementation of the proposed custom IP cores have been carried out on the RC1000 development board equipped with Virtex-2000E and the Low power FPGA (RC10 board) which is equipped with Spartan-3. It is worth mentioning that the proposed designs are platform independent and can be adopted/implemented on the most recent FPGAs. It would be nice to use the latest FPGA platforms feature (Virtex-6,7 and Spartan-6) and see their impacts; and

- Scope for improving the efficiency of deploying the power model presented in this work in iterative designs by using it to develop super-models incorporating a number of IP cores. The model based on estimation using software tools, it would be very interesting to build the model based real measurement data. This is a logical extension which is certain to achieve interesting outcomes; but at present has not been carried out due to time limitations.

7.3 Future Work

The work undertaken in this research project has concentrated on efficient implementation of some multiresolution algorithms (IP cores) for medical imaging applications. An in-depth high level power modelling methodology has also been applied in conjunction to a power aware design flow in order to optimise these cores. A set of objectives for the future include:

- *Further optimisation of the proposed multiresolution algorithms*:
  As researchers we can say that there is always place for more improvement almost in everything. In this case further improvement and optimisation can be done on all multiresolution algorithm IP cores which are not addressed in this work. In additions to medical imaging application the impact of the proposed IP cores on other image applications which contain many line and edge discontinuities is also an important objective for the future;

- *Cyclic convolution integration with proposed multiresolution algorithms*:
  This can be a very important objective in the future to integrate the proposed cyclic convolution into multiresolution algorithms for efficient computation and investigate its impacts. In addition, it would be interesting to evaluate the impact of MAC from Xilinx Corgen in the design (cyclic convolution) rather than coding the MAC using VHDL. It is worth noting that the proposed cyclic convolution implementation is coded in VHDL including MAC.

- *Further evaluation of the presented power model*:
  The presented power model is very successful in high level modelling of FPGA based IP cores. With the increasing tendency for IP core driven design, we believe this methodology holds a lot of promise for assisting in power aware design. However, a logical extension to this model will be the development of techniques to model a complete system comprising a number of IP cores, using individual core data. In addition, it would be very nice to evaluate the model further by collect data from real measurement of FPGA rather than using software estimation data;

- *Intelligent Optimisation Unit (IOU)*:
  Since power consumption on FPGA is influenced by a number of factors such as frequency, design density (number of interconnects), activity rates, logics and in-

terconnect structure of the specific FPGA, voltage levels and output load, a new environment can be introduced to the model to optimise even further. It consists of an IOU, control logic and memory. The IOU should predicts and optimise the

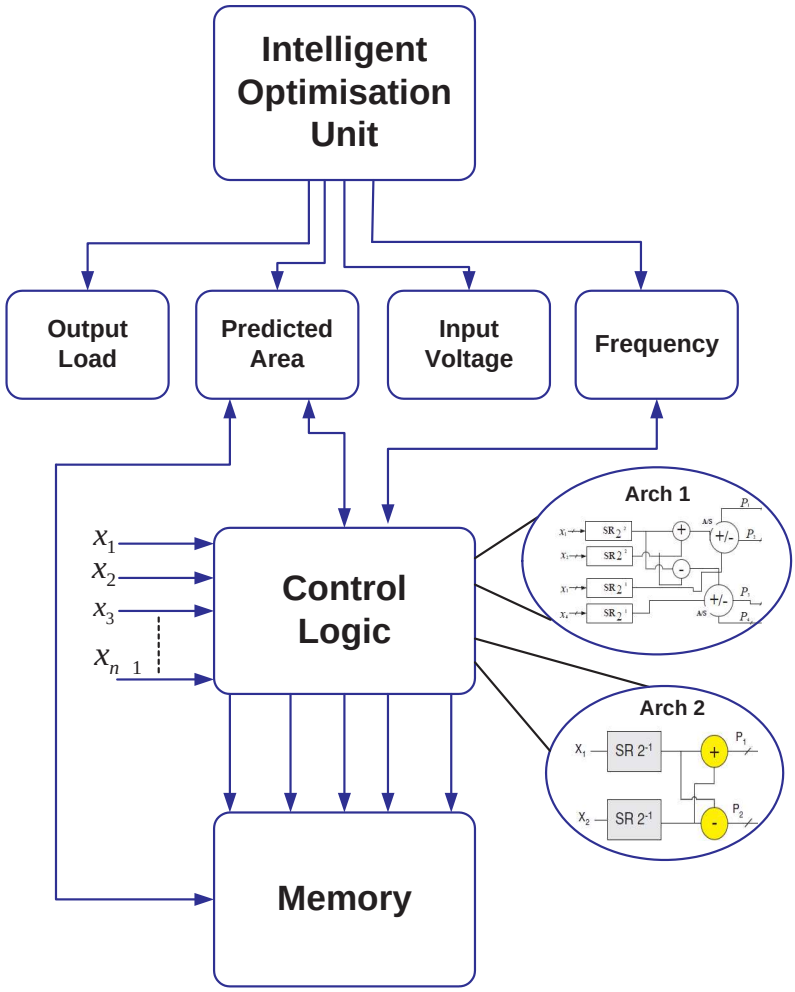


Figure 7.1: The steps involved to build the functional level modelling approach

power based on four parameters area (control logic and the memory), input voltage (depends on the activity rate), frequency (depends on the design itself) and output load. Providing sufficient detail to IOU the system should predict the power and give sufficient information about your design in terms of hardware implementation efficiency. It would be very interesting to see the outcome of this environment in future. The proposed intelligent optimisation unit environment is presented in Fig. 7.1.

- *Further validation on custom FPGA platforms*:

  It would be very interesting to evaluate the accuracy of the presented power model

on real FPGA development boards, However, in this case, only a generalised power model can be derived, as it is not possible to measure individual components of power such clock, signal, logic, etc. separately; and

- *Customised version of the IP cores*:
  Finally, it would be very nice to see customise version of the proposed cores and make it available for the end user to use them for their needs.

Bibliography

[1] R. Tessier and W. Burleson, "Reconfigurable Computing for Digital Signal Processing: A Survey," *J. VLSI Signal Process. Syst.*, vol. 28, pp. 7–27, May 2001.

[2] [Online]. Available: www.xilinx.com (05. 2009)

[3] T. Ahmed, P. D. Kundarewich, J. H. Anderson, B. L. Taylor, and R. Aggarwal, "Architecture-specific packing for virtex-5 FPGAs," *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays, Monterey, California, USA*, pp. 5–13, 2008.

[4] "Xilinx Virtex-5 65nm FPGA Family ," Xilinx, Articles. [Online]. Available: www.dsp-fpga.com/articles (22.10.2009)

[5] [Online]. Available: www. commons.wikimedia.org (04. 2010)

[6] P. D. Vecchio, Senior performance analyst, Software and Solutions Group, Intel Corp, Tech. Rep. [Online]. Available: www.developers.net/intelisnshowcase/view/127 (2009.04.04)

[7] C. Gao and S.-L. Lu, "Novel FPGA based Haar Classifier Face Detection Algorithm Acceleration," *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference*, pp. 373–378, 2008.

[8] A. M. Al-Haj, "An FPGA Based Parallel Distributed Arithmetic Implementation of the 1-D Discrete Wavelet Transform," *Signal Processing, The World Scientific and Engineering Academy and Society*, 2003.

[9] F. J. Diaz, A. M. Buron, and J. M. Solana, "Haar Wavelet Processor for Adaptive on-Line Image Compression," vol. 5837, no. 1. SPIE, 2005, pp. 204–212.

[10] Kang Sun, Xuezeng Pan, Zugen Liu and Tao Wu, "Design of A Reconfigurable Architecture for Discrete and Continuous Wavelet Transformsn," *Communication Technology, ICCT 06 International Conference on*, Nov 2006.

[11] A. Amira, A. Bouridane, P. Milligan, and M. Roula, "Novel FPGA Implementations of Walsh-Hadamard Transforms for Signal Processing," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 148, no. 6, pp. 377–383, 2001.

[12] A. Amira and S. Chandrasekaran, "Power Modeling and Efficient FPGA Implementation of FHT for Signal Processing," *IEEE Ttransaction On Very Large Scale Itegration (VLSI) Systems*, vol. 15, no. 3, pp. 286–296, March 2007.

[13] I. S. Uzun, "Design and FPGA Implementation of Matrix Transforms for Image and Video Processing," *PhD thesis, Queen's University of Belfast*, pp. 177–185, Sep 2006.

[14] S. Chandrasekaran, A. Amira, S. Minghua, and A. Bermak, "An efficient VLSI architecture and FPGA implementation of the Finite Ridgelet Transform," *Real-Time Image Proc, springer*, vol. 3, no. 15, p. 183 193, May 2008.

[15] C. A. Rahman and W. Badawy, "Architectures the Finite Radon Transform," *IEE Electronic Letters*, vol. 40, no. 15, pp. 931–932, July 2004.

[16] I. S. Uzun and A. Amira, "Design and FPGA Implementation of Finite Ridgelet Transform," *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 6, pp. 5826–5829, May 2005.

[17] J. Wisinger and R. Mahapatra, "FPGA Based Image Processing with the Curvelet Transform," Texas A & M University, TX, Tech Report TR-CS-2003-01-0, 2003.

[18] P. K.Meher and M. N. S. Swamy, "New Systolic Algorithm and Array Architecture for Prime-Length Discrete Sine Transform," *IEEE Trans. Circuits Syst. II*, vol. 54, pp. 262 –266, March 2007.

[19] C. Cheng and K. Parhi, "Hardware Efficient Fast DCT Based on Novel Cyclic Convolution Structures," *IEEE Transactions on Signal Processing*, vol. 54, no. 11, pp. 4419 –4434, Nov 2006.

[20] V. Sriram and D. Kearney, "A FPGA Implementation of Variable Kernel Convolution," *Parallel and Distributed Computing, Applications and Technologies, 2007. Eighth International Conference on Parallel and Distributed computing*, pp. 105–110, 2007.

[21] K. Mohammad and S. Agaian, "Efficient FPGA Implementation of Convolution," *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on System, Man, and Cybernetics*, pp. 3478–3483, 2009.

[22] N. Abdelli, A. Fouilliart, N. Mien, and E. Senn, "High-Level Power Estimation of FPGA," *ISIE 2007. IEEE International Symposium*, pp. 925 –930, June 2007.

[23] J. Laurent, N. Julien, E. Senn, and E. Martin, "Functional Level Power Analysis: An Efficient Approach for Modeling the Power Consumption of Complex Processors," *Proceedings of the conference on Design, automation and test in Europe*, p. 10666, 2004.

[24] K. K. Poon, S. J. E. Wilton, and A. Yan, "A Detailed Power Model for Field-Programmable Gate Arrays," *ACN Transactions on Design Automation of Electronic Systems*, vol. 10, no. 2, pp. 279–302, April 2005.

[25] L. Shang and N. K. Jha, "High-Level Power Modeling of CPLDs and FPGAs," *Proceedings of the International Conference on Computer Design*, pp. 46–51, September 2001.

[26] S. Chandrasekaran and A. Amira, "A New Behavioural Power Modelling Approach for FPGA based Custom Cores." IEEE Computer Society, 2007, pp. 350–357.

[27] V. Degalahal and T. Tuan, "Methodology for High Level Estimation of FPGA Power Consumption," *Proceedings of the Asia and South PacificDesign Automation Conference (ASP-DAC)*, vol. 1, pp. 657–660, Jan 2005.

[28] M. French, L. Wang, T. Anderson, and M. Wirthlin, "Post Synthesis Level Power Modeling of FPGAs," *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 281–282, April 2005.

[29] L. Zhong, S. Ravi, A. Raghunathan, and N. K. Jha, "Power Estimation for Cycle-accurate Functional Descriptions of Hardware," *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pp. 668– 675, November 2004.

[30] J. H. Anderson and F. N. Najm, "Power Estimation Techniques for FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 10, pp. 1015–1027, October 2004.

[31] F. Li, Y. Lin, L. He, D. Chen, and J. Cong, "Power Modeling and Characteristics of Field Programmable Gate Arrays," *IEEE Transactions nn Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 11, pp. 1712–1724, November 2005.

[32] "RC10 Platform Developers Kit," Celoxica Ltd., Manual, January 2008.

[33] "RC1000 Development Platform Product Brief," Celoxica Ltd., Datasheet v1.1, August 2002.

[34] "Virtex-4 Family Overview," Xilinx Inc., Tech. Rep. DS112 (v2.0), January 2007.

[35] "Handel-C Language Reference Manual," Metor Graphic Ltd., Tech. Rep., 2010.

[36] D. Taubman and M. Marcellin, "JPEG2000-Image Compression Fundamentals, Standards and Practice," *Kluwer Academic Publishers*, 2002.

[37] "Digital imaging and communications in medicine (dicom) supplement 61," *JPEG2000 Transfer Syntaxes*.

[38] A. J. Teh, P. Hobson, F. Ziliani, and J. Reichel, "Scalable Video Requirements for Surveillance Applications," *IEE Intelligent Distributed Surveillance Systems*, pp. 17–20, Feb 2004.

[39] S. Wu Bo and L. Shufang, "The Application of Wavelet Theory in Video Compression," *IEE Intelligent Distributed Surveillance Systems Antenna, Propagation and EMC Technologies for Wireless Communications*, vol. 2, pp. 1234–1236, Aug. 2005.

[40] D.-Z. Tian and M.-H. Ha, "Applications of Wavelet Transform in Medical Image Processing," *International Conference on Machine Learning and Cybernetics*, vol. 3, no. 1, pp. 1816–1821, August 2004.

[41] J. Starck, F. Murtagh, E. Candes, and D. Donoho, "Gray and Color Image Contrast Enhancement by the Curvelet Transform," *IEEE Transactions on Image Processing*, vol. 3, pp. 706–717, June 2003.

[42] M. Xing and Z. Bao, "High Resolutio ISAR Imaging of High Speed Moving Targets," *IEE Proceedings Radar, Sonar and Navigation*, vol. 152, pp. 58–67, April 2005.

[43] C. Poynton, "Digital Video and HDTV Algorithms and Interfaces (The Morgan Kaufmann Series in Computer Graphics)," *Morgan Kaufmann*, Jan. 2003.

[44] D. Lee, "JPEG 2000: Retrospective and New Developments," *Proceedings of the IEEE*, vol. 93, pp. 32–41, Jan 2005.

[45] C. Peisong and J. Woods, "Video Coding for Digital Cinema," *Proceedings of International Conference on Image Processing*, vol. 1, pp. 749–752, Sep 2002.

[46] M. N. Do and M. Vetterli, "Orthonormal Finite Ridgelet Transform for Image Compression," *Proceedings of the International Conference on Image Processing*, vol. 2, pp. 367–370, September 2000.

[47] E. J. Candes and D. L. Donoho, *Curves and Surfaces.* Vanderbilt University Press, 2000, ch. A Surprisingly Effective Nonadaptive Representation for Objects With Edges, pp. 105–120.

[48] E. Candes, "Ridgelets: Theory and Application," September 1998.

[49] R. Enzler, "The Current Status of Reconfigurable Computing," *Technical Report*, July 2000.

[50] M. Trope, "FPGAs: Past, Present, Future," *Electrical Engeneering and Computer Science, University of Kansas*, May 2004.

[51] B. L. Hutchings, "ASICs, Processors and Configurable Computing," *Proceedings of the* 30^{th} *Hawaii International Conference on System Sciences*, vol. 1, p. 719, January 1997.

[52] W. S. Carter, "The Future of Programmable Logic and its Impact on Digital System Design," *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 10–16, October 1990.

[53] B. Zahiri, "Structured ASICs: Opportunities and Challenges," *Proceedings of the International Conference on Computer Design*, pp. 404–409, October 2003.

[54] K. Wu and Y. Tsai, "Structured ASIC, Evolution or Revolution," *Proceedings of the International Symposium on Physical Design*, pp. 103–106, April 2004.

[55] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals: Architectures and Features.* IEEE Press, 1997.

[56] K. Rajan, K. S. Sangunni, and J. Ramakrishna, "Dual-DSP Systems for Signal and Image-Processing," *The EUROMICRO Journal on Microprocessing and Microsystems*, vol. 19, no. 9, pp. 556–560, 1993.

[57] T. Akiyama and H. Aono and K. Aoki and K. W. Ler and B. Wilson and T. Araki and T. Morishige and H. Takeno A. Sato and S. Nakatani and T Senoh, "MPEG2 video codec using image compression DSP," *IEEE Transactions on Consumer Electronics*, vol. 40, no. 3, pp. 466–472, August 1994.

[58] P. Donachy, "Design and Implementation of a High Level Image Processing Machine Using Reconfigurable Hardware," PhD Thesis, School of Computer Science, The Queen's University of Belfast, 1996.

[59] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, June 2002.

[60] [Online]. Available: www.mentor.com (2009)

[61] K. Benkrid, "Design and Implementation of a High Level FPGA Based Co-processor for Image and Video Processing," PhD Thesis, School of Computer Science, The Queen's University of Belfast, Tech. Rep., 2000.

[62] "Manual, "IEEE Standard VHDL Reference Manual"," IEEE Standard, Tech. Rep., 2000.

[63] Z. Navabi, "A High Level Language for Design and Modeling of Hardware," *Journal of System Software*, pp. 5–18, 1992.

[64] P. Moorby, "History of Verilog," *IEEE Design and Test of Computers*, vol. 9, no. 3, pp. 62–63, 1992.

[65] S. M. Loo, B. E. Wells, N. Freije, and J. Kulick, "Handel-C for Rapid Prototyping of VLSI Coprocessors for Real Time Systems," *Proceedings of the Thirty Fourth Southeastern Symposium on System Theory*, no. 3, pp. 6–10, March 2002.

[66] D. Sulik, M. Vasilko, D. Durackova, and P. Fuchs, "Design of a RISC Microcontroller Core in 48 Hours," *Embedded Systems Show*, May 2000.

[67] P. Voles, L. Holasek, and M. Vasilko, "ANSI C and Handel-C Based Rapid Prototyping Framework for Real Time Image Processing Algorithms," *Proceedings of the*

International Conference on Engineering of Reconfigurable Systems and Algorithms, Las Vegas, Nevada, USA,, Jun 2002.

[68] "Handel-C for Hardware Design," *White Paper*, vol. 9, Agust 2002.

[69] J. D. Crawford, "EDIF: A Mechanism for the Exchange of Design Information," *IEEE Design and Test of Computers*, vol. 2, no. 1, pp. 63–69, 1984.

[70] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, April 1965.

[71] G. Johnson, "At Los Alamos, Two Visions of Supercomputing," *The New York Times*, June 25 2002.

[72] A. Amira, A. Bouridane, P. Milligan, and A. Belatreche, "Design of efficient architectures for discrete orthogonal transforms using bit level systolic structures," *IEE Proceedings - Computers and Digital Techniques*, vol. 149, no. 1, pp. 17–24, January 2002.

[73] A. Amira, "An FPGA Based Parameterisable System for Discrete Hartley Transforms implementation," *Proceedings of the IEEE International Conference on Image Processing*, vol. 2, pp. 567–570, 2003.

[74] I. S. Uzun and A. Amira, "A FPGA-Based Parametrizable System for High-Resolution Frequency-Domain Image Filtering," *Journal Of Circuits Systems And Computers*, vol. 14, no. 5, pp. 895–922, February 2005.

[75] I. Uzun and A. Amira, "Real-time 2D wavelet transform implementation for HDTV compression," *Real-time imaging*, vol. 11, no. 2, pp. 151–165, April 2005.

[76] F. Bensaali and A. Amira, "Accelerating Colour Space Conversion on Reconfigurable Hardware," *Image and Vision Computing (Elsevier)*, vol. 23, no. 11, pp. 935–942, October 2005.

[77] A. Amira, "A Custom Coprocessor for Matrix Algorithm ," *PhD thesis, School of Computer Science, The Queen's University of Belfast*, 2001.

[78] "Virtex-E 1.8 V Field Programmable Gate Arrays," Xilinx Inc., Datasheet DS022-1 (v2.3), July 2002.

[79] F. Matus and J. Flusser, "Image Representation via a Finite Radon Transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 996–1006, October 1993.

[80] "Virtex-II platform FPGAs: Complete Data Sheet," Xilinx Inc., Datasheet DS031 (v3.3), June 2008.

[81] K. K. Poon, A. Yan, and S. J. E. Wilton, *Proceedings of the International Conference on Field Programmable Logic and Applications*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, September 2002, vol. 2438/2002, ch. A Flexible Power Model for FPGAs, pp. 312–321.

[82] T. Jiang, X. Tang, and P. Banerjee, "Macro-models for High Level Area and Power Estimationon on FPGAs," pp. 162–165, 2004.

[83] G. Strang, *"Introduction To Linear Algebra"*. Addison Wesley, 2003.

[84] P. Raviraj and M. Sanavullah, "The Modified 2D-Haar Wavelet Transformation in Image Compression," *Middle-East Journal of Scientific Research*, vol. 2, no. 2, pp. 73–78, 2007.

[85] S. A. White, "Applications of distributed arithmetic to digital signal processing : Tutorial review," *ASSP Magazine*, vol. 6, p. 419, 1989.

[86] K. P. Lim and A. B. Premkumar, "A Modular Approach to the Computation of Convolution Sum Using Distributed Arithmetic Principles," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, p. 9296, 1990.

[87] G. R. Goslin, "A Guide to Using Field Programmable Gate Arrays FPGAs for Application Specific Digital Signal Processing Performance," *Proceedings of SPIE in High-Speed Computing, Digital Signal Processing*, pp. 321–331, 1996.

[88] L.-W. Chang and M.-C. Wu, "A Bit Level Systolic Array for Walsh-Hadamard Transforms," *Signal Processing*, vol. 31, no. 3, pp. 341–347, April 1993.

[89] S. S. Nayak and P. K. Meher, "High Throughput VLSI Implementation Of Discrete Orthogonal Transforms Using Bit-Level Vector-Matrix Multiplier," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, no. 5, pp. 655–658, May 1999.

[90] "Virtex-5 Family Overview," Xilinx Inc., Datasheet, February 2009.

[91] "Spartan-3L Low Power FPGA Family," Xilinx Inc., Tech. Rep. DS313 (v1.1), September 2005.

[92] A. Amira, S. Chandrasekaran, "Power Modelling and Efficient FPGA Implementation of FHT for Signal Processing," *IEEE Transactions on VLSI System*, pp. 286–295, 2007.

[93] Joseph D. Bronzino, "The biomedical engineering handbook, Medical devices and systems," *CRC Press*, vol. 2, 2006.

[94] J. Beutal, H. Kundel and R. Van Metter, "Handbook of Medical Imaging," *Physics and Psychophysics, SPIE Press*, 2006.

[95] M.S. Sharif, A. N. Sazish, and A. Amira, "An Efficient Algorithm and Architecture for Medical Image Segmentation and Tumour Detection," *IEEE Biomedical Circuits and Systems Conference (BIOCAS)*, November 2008.

[96] A. Sazish, M. Sharif, and A. Amira, "Hardware Implementation and Power Analysis of HWT for Medical Imaging," *16th IEEE International Conference on Electronics, Circuits, and Systems, 2009 ICECS 2009.*, pp. 775 –778, Dec. 2009.

[97] [Online]. Available: www.mathworks.com/products/matlab (03. 2010)

[98] R. J. Petersen and B. Hutchings, "An Assessment of the Suitability of FPGA-Based Systems for Use in Digital Signal Processing," *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications*, pp. 293–302, 1995.

[99] T. J. Callahan and J. Wawrzynek, "Instruction-Level Parallelism for Reconfigurable Computing," *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications, From FPGAs to Computing Paradigm*, pp. 248–257, 1998.

[100] N. Rollins and M. Wirthlin, "Reducing energy in FPGA multipliers through glitch reduction," *Proceedings of the* 7^{th} *Annual International Conference on Military Applications of Programmable Logic Devices*, September 2005.

[101] S. J. E. Wilton, S.-S. Ang, and W. Luk, "The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays," *Lecture Notes in Computer Science*, pp. 719–728, 2004.

[102] E. D. Bolker, "The finite Radon transform," S. Helgason, R. L. Bryant, V. Guillemin, , and R. O. Wells, Jr, Eds., 1987, vol. 63, pp. 27–50.

[103] P. Toft, "The Radon Transform - Theory and Implementation," Ph.D. Thesis, Informatics and Mathematical Modelling, Technical University of Denmark, 1996.

[104] J. I. Guo, C. M. Liu, and C. W. Jen., "A new array architecture for prime-length discrete cosine transform," *IEEE Trans.Signal Processing*, vol. 41, p. 436 442, January 1993.

[105] D. F. Chiper, M. N. S. Swamy, M. O. Ahmad, and T. Stouraitis, "A systolic array architecture for the discrete sine transform," vol. 50, pp. 2347 –2354, Sept 2002.

[106] P. K. Meher, J. C. Patra, and M. N. S. Swamy, "Highthroughput Memory Based Architecture for DHT Using a New Convolutional Formulation," *IEEE Trans. Circuits Syst. II*, vol. 54, pp. 606 –610, July 2007.

[107] H. C. Chen, J. I. Guo, T. S. Chang, and C.-W. Jen, " A Memory Efficient Realization of Cyclic Convolution and Its Application to Discrete Cosine Transform," *IEEE Trans. Circuits Syst for Video Technol*, vol. 15, pp. 445 –453, March 2005.

[108] D. F. Chiper, M. N. S. Swamy, M. O. Ahmad, and T. Stouraitis, "Systolic Algorithms and A Memory Based Design approach for a unified architecture for the computation of DCT/DST/IDCT/IDST," *IEEE Trans. Circuits Syst-I*, vol. 52, pp. 1125 –1137, June 2005.

[109] P. K. Meher, "Systolic designs for DCT using a lowcomplexity concurrent convolutional formulation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 16, pp. 041 –1050, Sep 2006.

[110] ——, "Efficient Systolization of Cyclic Convolution for Systolic Implementation of Sinusoidal Transforms," *Application Specific Systems Architectures and Processors conference ASAP 2008*, pp. 97 – 101, July 2008.

[111] R. C. Agarwal and J. W. Cooley, "New algorithms for Digital Convolution," *IEEE Trans. Acoust. Speech Signal Process*, vol. 25, pp. 392 –410, Oct 1977.

[112] A. C. Atoche, D. T. Roman, and Y. Shkvarko, "Reconfigurable Architecture of Systolic Array Processors for Real Time Remote Sensing Image Enhance-

ment/Reconstruction," *WSEAS TRANSACTIONS on SIGNAL PROCESSING*, vol. 5, pp. 293 –303, August 2009.

[113] L. Stok and J. Cohn , "There is Life Left in ASICs," *IEEE Int.Symp. Physical Design*, pp. 48–50, 2003.

[114] L. Shang, A. Kaviani, and K. Bathala, "Dynamic Power Consumption in the Virtex-II FPGA Family," *in Proc. ACM Int. Symp. Field-Programmable Gate-Arrays*, p. 157164, 2002.

[115] S. Chandrasekaran and A. Amira, "A New Behavioural Power Modelling Approach for FPGA based Custom Cores," *Conference on Adaptive Hardware and Systems, NASA/ESA Los Alamitos, CA, USA*, pp. 350–357, 2007.

[116] [Online]. Available: www.nlreg.com (2008)

[117] T. Osmulski, J. T. Muehring, B. Veale, J. M. West, H. Li, S. Vanichayobon, S.-H. Ko, J. K. Antonio, and S. K. Dhall, "A Probabilistic Power Prediction Tool for the Xilinx 4000-Series FPGA," *Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, pp. 776–783, 2000.

[118] H. G. Lee, S. Nam, and N. Chang, "Cycle-accurate energy measurement and high-level energy characterization of FPGAs," *Proceedings of the Fourth International Symposium on Quality Electronic Design*, pp. 267–272, March 2003.

[119] F. Bensaali, "Accelerating Matrix Product on Reconfigurable Hardware for Image Processing Applications," Phd Thesis, School of Computer Science, The Queen's University of Belfast, May 2005.

[120] "FPGA Co-Processing Solutions for High-Performance Signal Processing Applications," Altera Corporation, Altera Corporation, 101 Innovation Drive, San Jose, California 95134, USA, White Paper WP-041905-1.1, May, online: http://www.altera.com/literature/.

[121] J. C. Chen, K. M. Sivalingam, P. Agrawal, and R. Acharya, "Scheduling Multimedia Services in a Low-Power MAC for Wireless and Mobile ATM Networks," *IEEE Transactions on Multimedia*, vol. 1, no. 2, pp. 187–201, June 1999.

[122] A. Sazish and A. Amira, "An Efficient Architecture for HWT Using Sparse Matrix Factorisation and DA Principles," *IEEE Asia Pacific Conference on Circuits and Systems, 2008. APCCAS 2008.*, pp. 1308 –1311, Dec 2008.

[123] A. Sazish, S. Chandrasekaran, and A. Amira, "Efficient Systolic Architecture and Power Modeling for Finite Ridgelet Transform," *IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops), 2009*, pp. 821 –827, Oct. 2009.

[124] I. Page and W. Luk, "Compiling Occam into Field-Programmable Gate Arrays," *FPGAs, Oxford Workshop on Field Programmable Logic and Applications*, pp. 271–283, 1991.

[125] *DK Design Suit*, Version 5.2 ed., Metoe Graphic Ltd, 2010.

[126] "Xpower Tutorial: FPGA Design," Xilinx, Tech. Rep., July 2002.

Appendix A

FPGA Prototyping Board

In this thesis two FPGA boards RC10 and RC1000 are used to prototype the proposed designs. RC10 board is fitted with Xilinx Spartan 3 XC3S1500L-4-FG320 FPGA and is packaged with a set of comprehensive support libraries intended for use with Celoxica's suite of ESL design tools including the DK Design Suite and PixelStreams image and video processing library [32]. The RC10 is packed with a powerful set of I/O features including 2 high-speed ADC channels, VGA video output, 1-bit DAC audio outputs, CAN bus and a CMOS camera connector. A high-speed USB 2.0 interface allows high data-rate communication between PC host programs and FPGA applications on the board. RC10 is a portable board which can be access through USB, beside the FPGA it has other additional features including 8 LEDs, 2 seven-segment displays, PS/2 keyboard and mouse sockets, and a 5-way mini joystick which can be used in conjunction with FPGA. RC10 also provides a user interface (FTU3) to upload or access the FPGA through USB. RC10 Board expansion capabilities include a 50-way expansion socket providing 33 data I/Os, 2 clock pins as well as +12V, +5V and +3.3V power supplies, a 4 way LVDS connector also suitable for driving a TFT screen, and connections for 4 standard servos. Fig. A.1 shows the RC10 and its features.

A.0.1 Spartan 3 FPGA

The Spartan-3L family architecture consists of five fundamental programmable functional elements [32]:

- Configurable Logic Blocks (CLBs) contain RAM-based Look-Up Tables (LUTs) to implement logic and storage elements that can be used as flip-flops or latches. CLBs

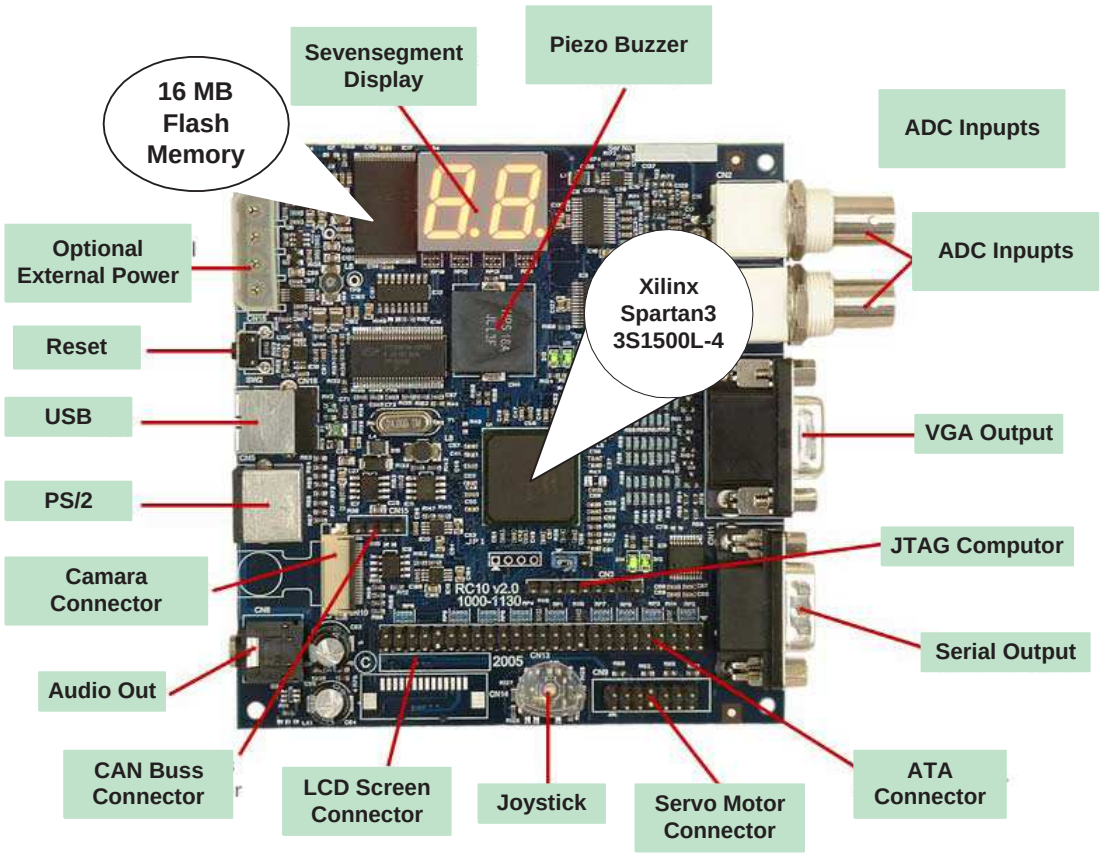


Figure A.1: RC10 board with Xilinx Spartan

can be programmed to perform a wide variety of logical functions as well as to store data.

- Input/Output Blocks (IOBs), Spartan 3l family control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Twenty-six different signal standards, including eight high-performance differential standards, are available. Double Data-Rate (DDR) registers are included. The Digitally Controlled Impedance (DCI) feature provides automatic on-chip terminations, simplifying board designs.

- Block RAM provides data storage in the form of 18 Kbit dual-port blocks.

- Multiplier Blocks accept two 18-bit binary numbers as inputs and calculate the product.

- Digital Clock Manager (DCM) Blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase-shifting clock signals. The block diagram of Spartan 3l is shown in Fig. A.2

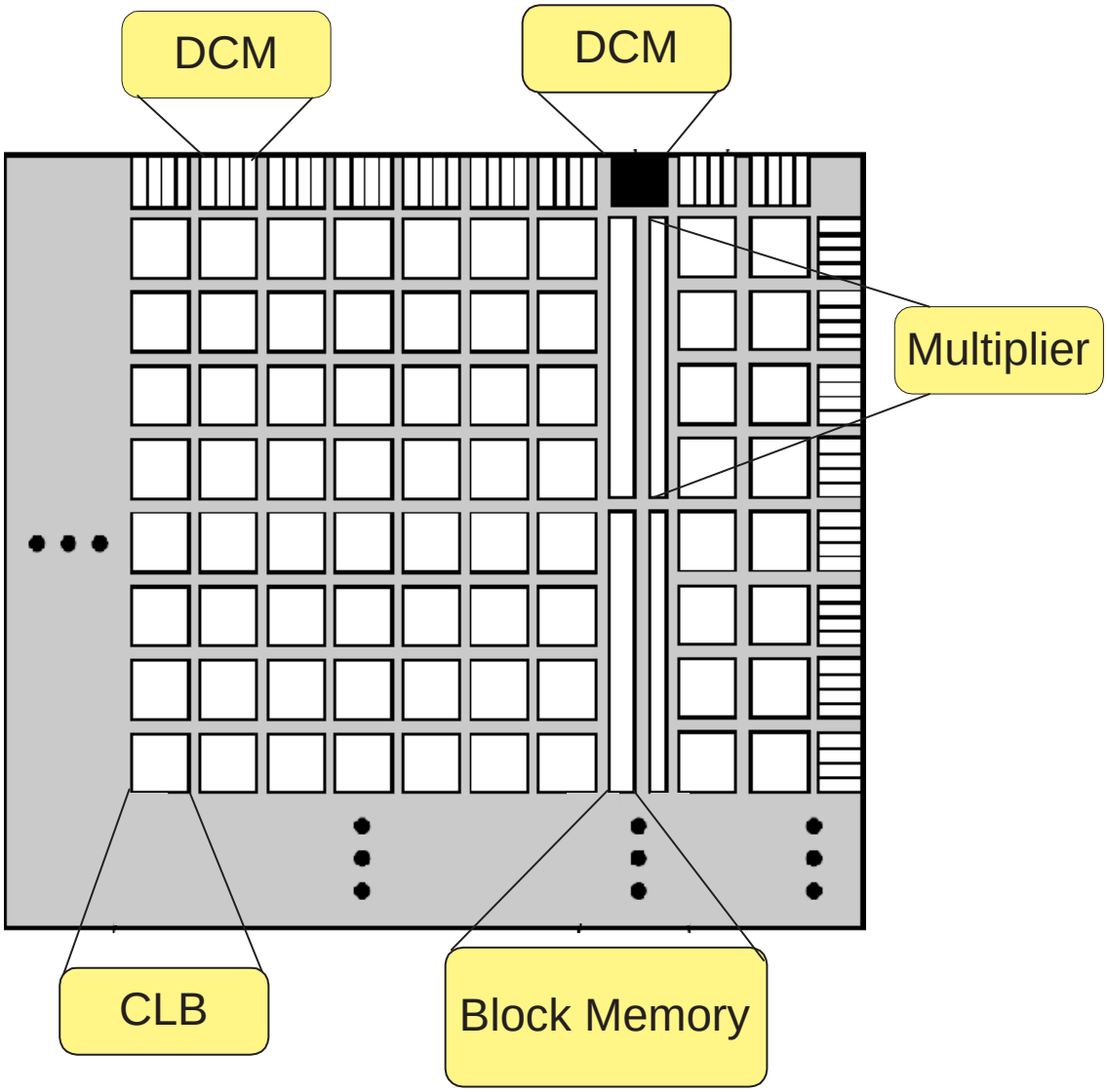


Figure A.2: Xilinx Spartan 3 block diagram [32]

A.0.2 RC1000 FPGA Board

RC1000 is a standard PCI bus plug-in card for PCs equipped with a Xilinx XCV2000E-6 Virtex-E FPGA chip. It has 8 MB of SRAM directly connected to the FPGA in four 32 bit wide memory banks. The memory is also visible to the host CPU across the PCI bus as if it was normal memory. Each of the 4 banks may be granted to either the host CPU or the FPGA at any time. Data can therefore be shared between the FPGA and host CPU by placing it in the SRAM on the board. It is then accessible to the FPGA directly and to the host CPU either by Direct Memory Access (DMA) transfers across the PCI bus or simply as a virtual ad- dress. High speed DMA, data buffering and clock speed control

make it suitable for high speed cryptographic applications. The actual Rc1000 board is shown in Fig. A.3 and the functional block for RC1000 is shown in Fig. A.4

The FPGA has two of its pins connected to the clocks. One pin is connected to either a programmable clock or an external clock. The programmable clocks are programmed by the host PC, and have frequency range of 400 KHz to 100 MHz [33].

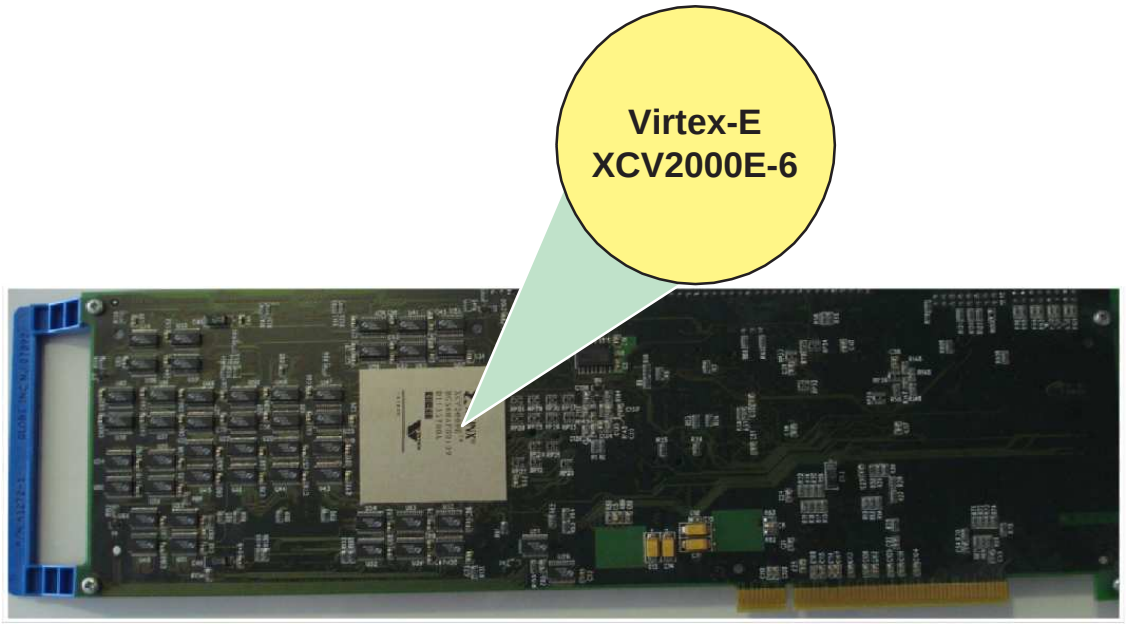


Figure A.3: Xilinx RC1000 board

The RC1000 board is supported with a macro library (the PP1000 library,) that simplifies the process of initialising and talking to the hardware. This library comprises driver functions with the following functionality:

- Initialisation and selection of a board
- Handling of FPGA configuration files
- Data transfer between PC and the RC1000 board
- Function to help with error checking and debugging these library functions can be included in a C or C++ program that runs on the host PC and performs data transfer via the PCI bus [35]

DMA, data buffering and clock speed control make the RC1000 suitable for high-speed applications.

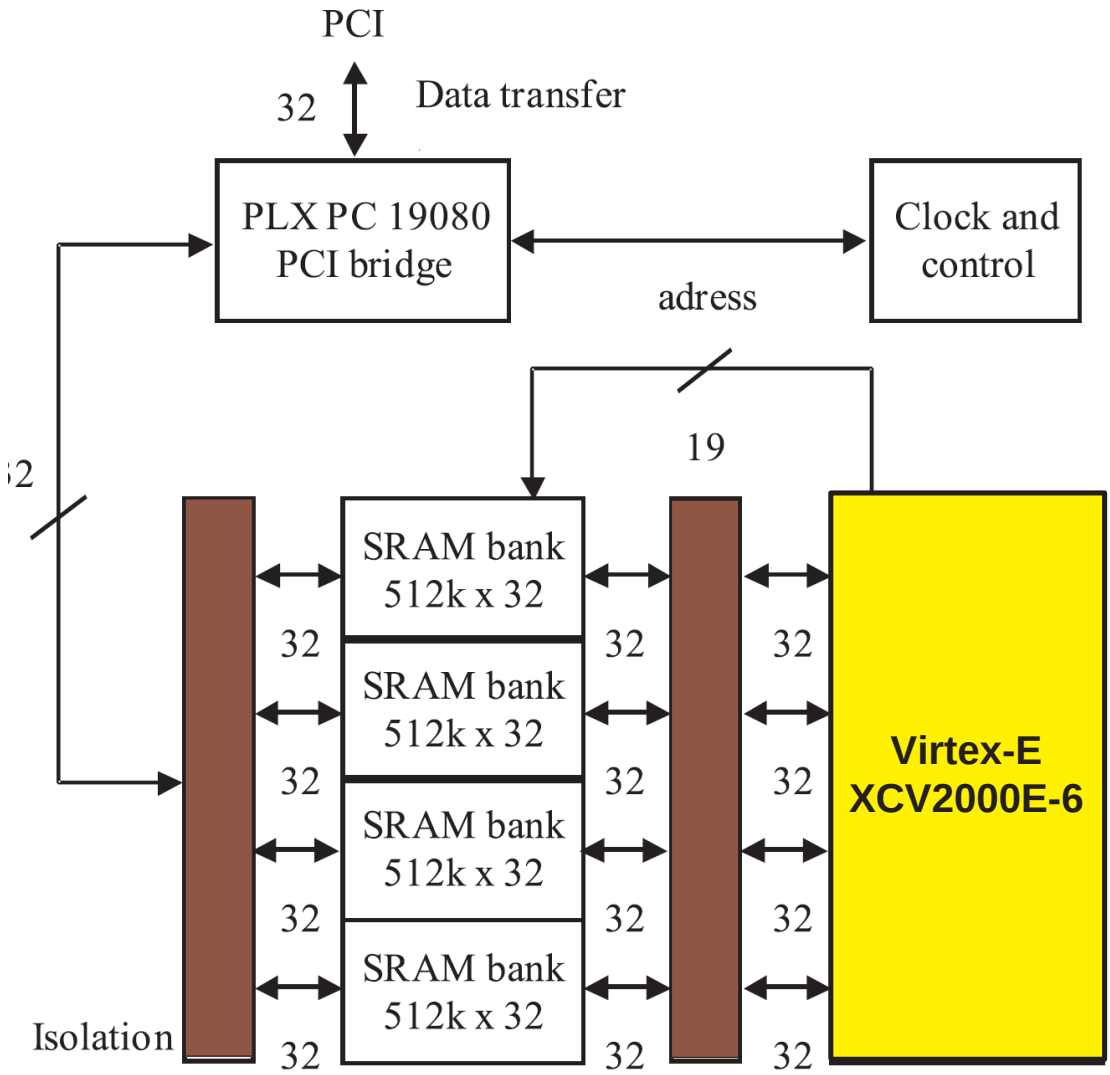


Figure A.4: Xilinx RC100 block diagram [33]

A.1 Other FPGA Devices Used in This Research

In this research a variety of FPGA devices have been used in order to implement the proposed architectures for evaluation and comparison purpose. The choice of the device used depends on previous works and analysis factors:

A.1.1 Virtex-5

The Virtex-5 family provides the newest most powerful features in the FPGA market. Using the second generation Advanced Silicon Modular Block (ASMBL) column-based

architecture. Each Virtex-5 platform contains a different ratio of features to address the needs of a wide variety of advanced logic designs. In addition Virtex-5 FPGAs contain many hard-IP system level blocks, including powerful 36-Kbit block RAM/FIFOs, second generation 25x18 DSP slices, SelectIO technology with built in digitally controlled impedance, system monitor functionality, enhanced clock management tiles with integrated Digital Clock Managers (DCM) and phase-locked-loop (PLL) clock generators, and advanced configuration options. and high-performance PowerPC 440 microprocessor embedded blocks. These features allow advanced logic designers to build the highest levels of performance and functionality into their FPGA-based systems. Virtex-5 FPGAs offer the best solution for addressing the needs of high-performance logic designers, high-performance DSP designers, and high-performance embedded systems designers with unprecedented logic, DSP, hard/soft microprocessor, and connectivity capabilities. The Virtex-5 LXT, SXT, TXT, and FXT platforms include advanced high-speed serial connectivity and link/transaction layer capability [90]. A simplified diagram of Xilinx Virtex-5 FPGA slice is shown in Fig. A.5 and ML505 evaluation platform which equipped with Xilinx Virtex-5 LXT FPGA is shown in Fig. **??**. The most important features of Virtex-5 is listed below:

- Most advanced, high-performance, optimal-utilization, FPGA fabric, real 6-input Look-Up Table (LUT) technology, 64-bit distributed RAM option, advanced DSP48E slices
- Powerful clock management tile (CMT) clocking
- 36-Kbit block RAM/FIFOs
- High-performance parallel SelectIO technology
- System Monitoring capability on all devices
- Integrated Endpoint blocks for PCI Express Designs

A.1.2 Virtex-4

Virtex-4 family from Xilinx greatly enhances programmable logic design capabilities, making it a powerful alternative to ASIC technology. Virtex-4 FPGAs comprise three platform families LX, FX, and SX offering multiple feature choices and combinations to address all

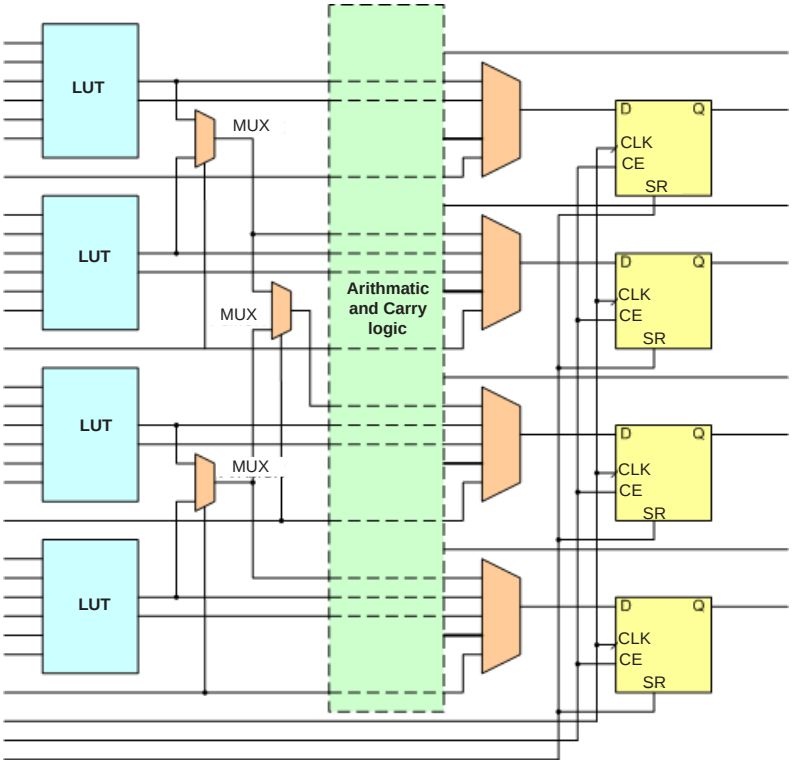


Figure A.5: A simplified diagram of a Xilinx Virtex-5 FPGA slice [4]

complex applications. The wide array of Virtex-4 FPGA hard-IP core blocks includes the PowerPC processors (with a new APU interface), MACs, 622 Mb/s to 6.5 Gb/s serial transceivers, dedicated DSP slices, high speed clock management circuitry, and source synchronous interface blocks. The basic Virtex-4 FPGA building blocks are enhancements of those found in the popular Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X product families, so previous-generation designs are upward compatible. Virtex-4 devices are produced on a state of the art 90nm copper process using 300 mm (12-inch) wafer technology [34]. A simplified diagram of a Xilinx Virtex-4 FPGA slice is shown in Fig A.6. The important features of Vitex-4 are listed below:

- Xtreme DSP Slice 18 x 18, twos complement, signed Multiplier.
- Built-in Accumulator (48-bit) and Adder/Subtracter.
- Smart RAM Memory Hierarchy (distributed RAM and dual-port 18-Kbit RAM blocks)
- IBM PowerPC RISC Processor Core
- Multiple Tri-Mode Ethernet MACs

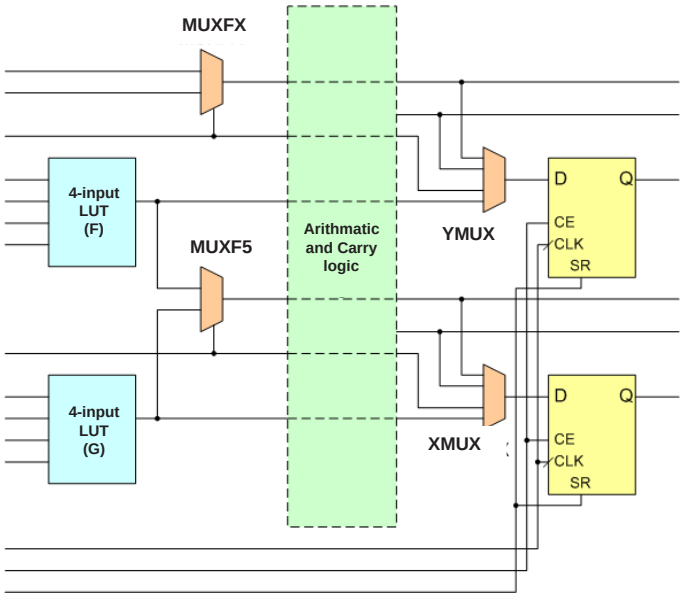


Figure A.6: A simplified diagram of a Xilinx Virtex-4 FPGA slice [34]

Appendix B

Tools and Software Packages

There are many tools and software packages which are used to program FPGAs, in this research DK suit and ISE are used to program the proposed designs on FPGAs, more details about these two tools are explored in the following sections.

B.1 Handel-C

Handel-C is a high level programming language which targets low-level hardware, most commonly used in the programming of FPGAs. It is a rich subset of C, with non-standard extensions to control hardware instantiation with an emphasis on parallelism. Fig B.1 shows the most important additional features between Hadel-C and ANSI-C.

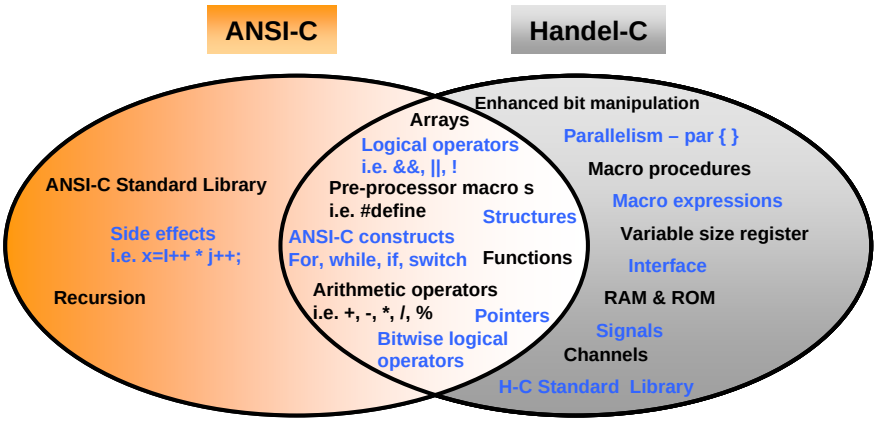


Figure B.1: Handel-C/ANSI-C comparison [35]

Handel-C is to hardware design what the first high level programming languages were to programming CPUs. Unlike many other design languages that target a specific architec-

ture Handel-C can be compiled to a number of design languages and then synthesised to the corresponding hardware. This frees developers to concentrate on the programming task at hand rather than the idiosyncrasies of a specific design language and architecture. Research into the field of hardware compilation for FPGAs started in 1991, when Ian Page and Wayne Luk (Hardware Compilation Group at at the Programming Research Group (PRG) within the Oxford University Computing Laboratory (OUCL)), developed a compiler that transformed a subset of Occam into a netlist suitable for loading onto an FPGA [124].

The technology developed at Oxford was spun off to mature as a cornerstone product for Embedded Solutions Limited (ESL) in 1996. ESL was renamed Celoxica in September 2000. Handel-C was adopted by many university hardware research groups after its release by ESL, as a result was able to established itself as a hardware design tool of choice within the academic community, especially in the United Kingdom. In early 2006, Celoxica's ESL business was acquired by Catalytic, a startup selling a MATLAB to C tool. Soon thereafter, Celoxica and Catalytic merged to form Agility, which developed and sold, among other products, ESL tools supporting Handel-C. In January 2009, Mentor Graphics acquired Agility's C synthesis assets. Handel-C aimed at compiling high level algorithms directly into gate level hardware. In order to support the use of the language the vendor also supplies a graphical design environment called DK (Design Kit) (Fig. B.2) that incorporates simulator, debugger, compiler and implementation generation, in EDIF, VHDL or Verilog [125].

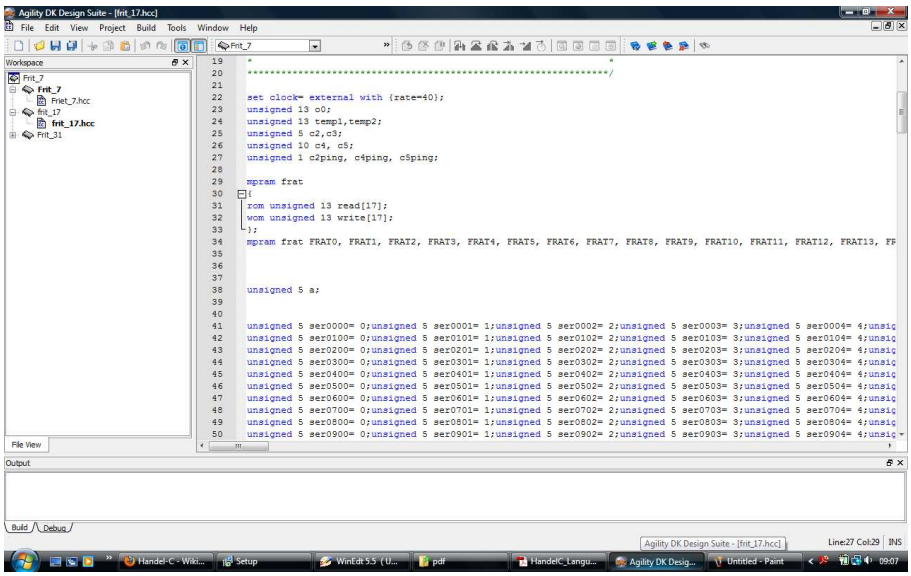
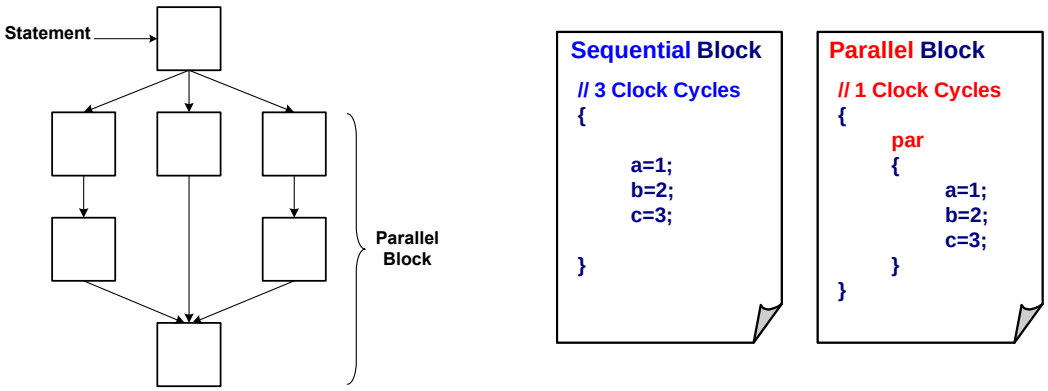


Figure B.2: The DK design synthesis tool

B.1.1 Parallel Hardware Generation

The target of the Handel-C compiler is low-level hardware. This means that you get massive performance benefits by using parallelism. It is essential for writing efficient programs to instruct the compiler to build hardware to execute statements in parallel. Handel-C parallelism is true parallelism, not the time-sliced parallelism familiar from general-purpose computers. When instructed to execute two instructions in parallel, those two instructions will be executed at exactly the same instant in time by two separate pieces of hardware. When a parallel block is encountered, execution flow splits at the start of the parallel block and each branch of the block executes simultaneously. Execution flow then re-joins at the end of the block when all branches have completed. Any branches that complete early are forced to wait for the slowest branch before continuing. Fig B.3 [35].



(a) Parallel branch execution flow (b) *PAR* construct example

Figure B.3: The *PAR* construct [35]

B.1.2 Channel Communications

Channels provide a link between branches executing in parallel. One parallel branch outputs data onto the channel and the other branch reads data from the channel. Channels can be constructed with and without FIFO capacities.

- Channels constructed as FIFOs, a channel can be constructed as a FIFO queue. In this case, the data is written to the head of the FIFO is read from the tail. If the FIFO is full, write blocks until an element is read from the FIFO. If the FIFO is empty, read blocks until there is data ready to be read

- Channels constructed without FIFO capacity, these channels provide synchronization between parallel branches because the data transfer can only complete when both the transmitter and the receiver are ready. If one side is not ready, the other must wait.

In Fig B.4, the channel is shown transferring data from the left branch to the right branch. If the left branch reaches point a before the right branch reaches point b, the left branch waits at point *a* until the right branch reaches point *a* [35].

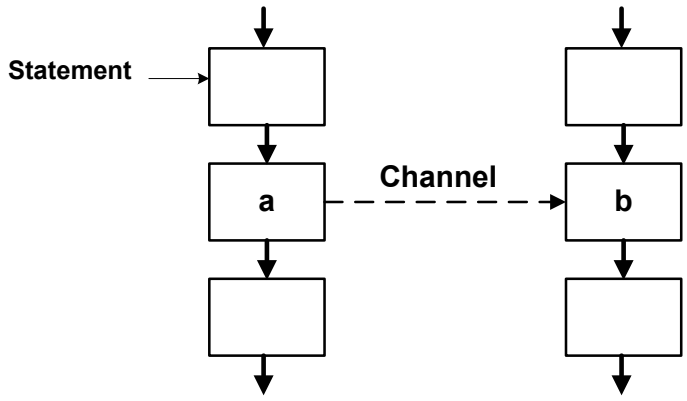


Figure B.4: Channel communication [35]

B.1.3 Memory

RAMs and ROMs can be implemented directly using the keywords *ram* and *rom* respectively. Handel-C allows access to a number of different types of RAM:

- Distributed RAM, which is implemented in look-up tables in the logic blocks of the FPGA
- Block RAM, which is available on certain chips, can be identified by specifying the *block* parameter in conjunction with the *ram* keyword
- Off-chip RAM [35]

B.2 Xilinx-ISE

ISE is a software tool produced by Xilinx for synthesis and analysis of HDL designs, which enables the developer to synthesize (”compile”) their designs, perform timing analysis,

examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Through the Project Navigator interface, you can access all of the design entry and design implementation tools. Fig.B.5 shows the snapshot of ISE 9.1i project navigator. The different design entry steps in ISE are described in the following subsections.

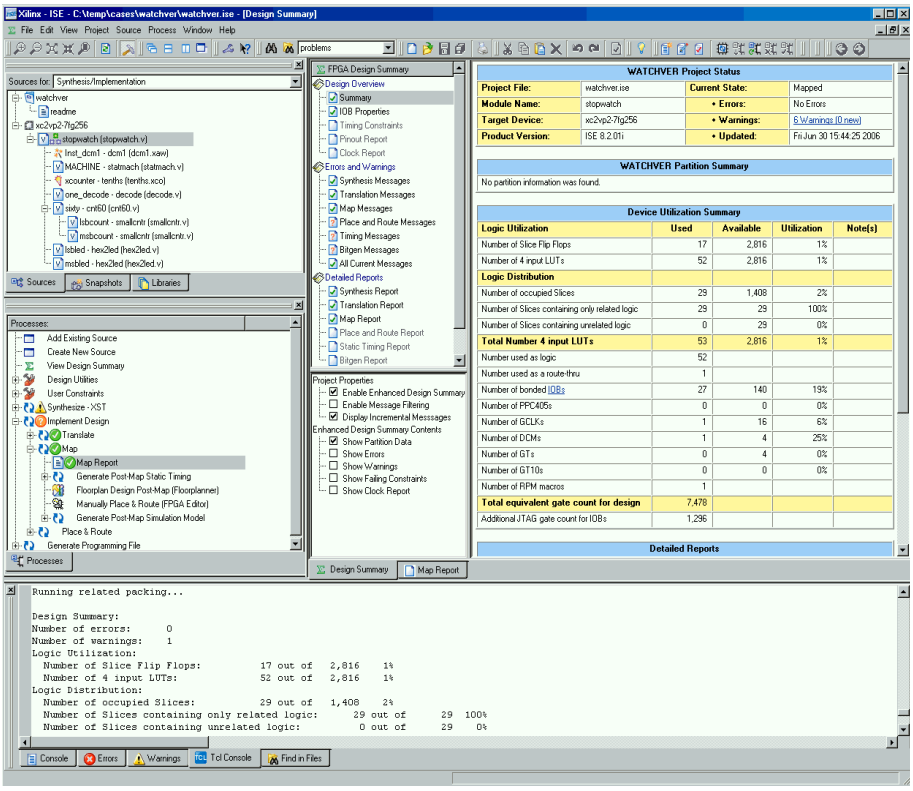


Figure B.5: ISE Project navigator display window

B.2.1 ISE Translation

The translate process merges all of the input netlists and design constraints and outputs a Xilinx native generic database (NGD) file, which describes the logical design reduced to Xilinx primitives. During translation, the NGD build program performs the following functions. Converts input design netlists and writes results to a single merged NGD netlist and adds the User Constraints File (UCF) to the merged netlist.

B.2.2 ISE Timing Constraints

Timing constraints are typically specified globally but can also be specified for individual paths. Global constraints include period constraints for each clock (PERIOD), setup times for each input (OFFSET-IN), and clock-to-out constraints for each output (OFFSET-OUT). You can enter timing constraints using the Create Timing Constraints process in project navigator. This creates a text-based UCF, which is used during implementation. You can enter timing constraints for synthesis in a separate XCF file. More information on entering timing constraints is provided in [2]. Results for the timing constraints are automatically reported after implementation, and are also available from the ISE Design Summary. To analyze the results of your timing specifications, use Timing Analyzer or the command line tools TRACE (Timing Reporter and Circuit Evaluator) for FPGAs.

B.2.3 Placement Constraints

For FPGAs, you can specify placement constraints for each type of logic element, such as flip-flops, ROMs and RAMs, FMAPs, BUFTs, CLBs, IOBs, I/Os, and global buffers in FPGA designs. Individual logic gates, such as AND and OR gates, are mapped into CLB function generators before the constraints are read and cannot be constrained. However, if gates are represented by an FMAP symbol, you can use a placement constraint on that symbol.

B.2.4 Synthesis Constraints

Synthesis constraints instruct the synthesis tool to perform specific operations. When using XST for synthesis, synthesis constraints control how XST processes and implements FPGA resources. Synthesis constraints also allow control of register duplication (REGISTER-DUPLICATION) and fanout control (MAX-FANOUT) [2] during global timing optimization. To give XST specific targets during global optimization, you can enter timing constraints for synthesis in the XCF file.

B.2.5 Place & Route

Place & Route (PAR) are composed of two steps, placement and routing. The first step, placement, involves deciding where to place all electronic components, circuitry, and logic elements in a generally limited amount of space. This is followed by routing, which decides

the exact design of all the wires needed to connect the placed components. This step must implement all the desired connections while following the rules and limitations of the manufacturing process. The PAR can be verified using ISE FPGA Editor. The FPGA Editor reads and writes Native Circuit Description (NCD) files, Nativc Macro Circuit (NMC) files and Physical Constraints Files (PCF). It performs the following tasks:

- Place and route critical components before running the automatic place-and-route tools;
- Finish placement and routing if the routing program does not completely route your design;
- Add probes to the design to examine the signal states of the targeted device;
- View and change the nets connected to the capture units of an Integrated Logic Analyser (ILA) core in the design;
- Run the BitGen program and download the resulting bitstream file to the targeted device; and
- View and change the nets connected to the capture units of an Integrated Logic Analyser (ILA) core in your design.

A snapshot of the FPGA editor is shown in Fig. B.6.

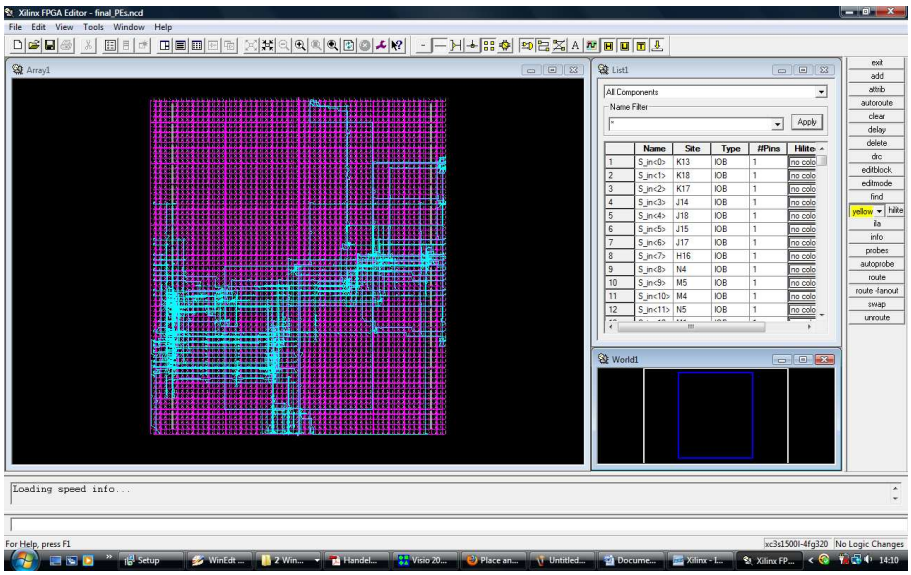


Figure B.6: FPGA Editor showing the place and route of a design

B.2.6 Core Generator

The CORE Generator System is a design tool that delivers parameterized COREs optimized for Xilinx FPGAs. It provides you with a catalog of ready-made functions ranging in complexity from simple arithmetic operators such as adders, accumulators, and multipliers, to system-level building blocks such as filters, transforms, FIFOs, and memories [2]. For each core it generates, the CORE Generator System produces an Electronic Data Interchange Format (EDIF) netlist (EDN file), a Verilog template (VEO) file with a Verilog (V) wrapper file, and/or a VHDL template (VHO) file with a VHDL (VHD) wrapper file. It may also create one or more NGC and NDF files. NGC files are produced for certain cores only.

The Electronic Data Netlist (EDN) and NGC files contain the information required to implement the module in a Xilinx FPGA. Since NGC files are in binary format, ASCII NDF files may also be produced to communicate resource and timing information for NGC files to 3rd party synthesis tools. The ASY and XSF symbol information files allow you to integrate the CORE Generator module into a schematic design for Mentor or ISE tools. VEO and VHO template files contain code that can be used as a model for instantiating a CORE Generator module in a Verilog or VHDL design. Finally, V and VHD wrapper files are provided to support functional simulation. These files contain simulation model customization data that is passed to a parameterized simulation model for the core. In the case of Verilog designs, the V wrapper file also provides the port information required to integrate the core into a Verilog design for synthesis the over all design flow of Xilinx Core Generator is shown in Fig. B.7.

B.2.7 XPower Estimation

XPower calculates power based on dynamic power consumption in CMOS circuits which is primarily due to switching activity. Each element (LUT, FF, BRAM, routing segment) that can switch has a capacitance model associated with it. Clock signals and primary input signals are assigned specific frequencies by the user. Synchronous elements are assigned activity (or toggle) rates relative to their associated clock. Activity rates for DLL and BUFG outputs are set relative to their input clock. User-supplied activity rates combine with device-specific capacitance, static power, and other data to produce a power estimate for a design. The accuracy of the switching activity data is crucial in obtaining

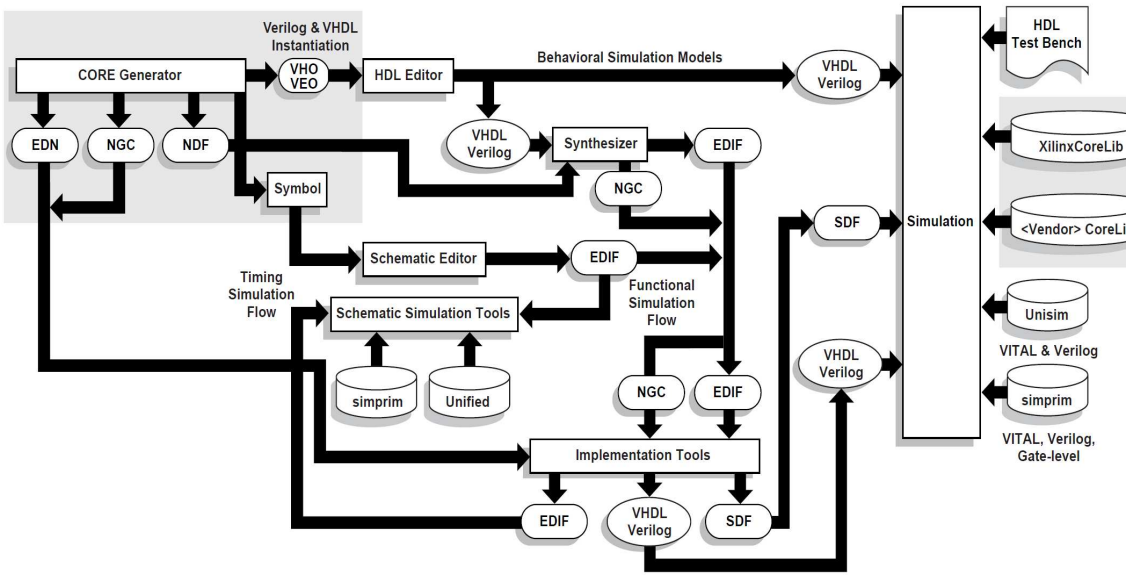


Figure B.7: The design flow of Xilinx Core Generator

an accurate estimate of power consumption [126]. The ISE Xpower interface is shown in Fig. B.8.

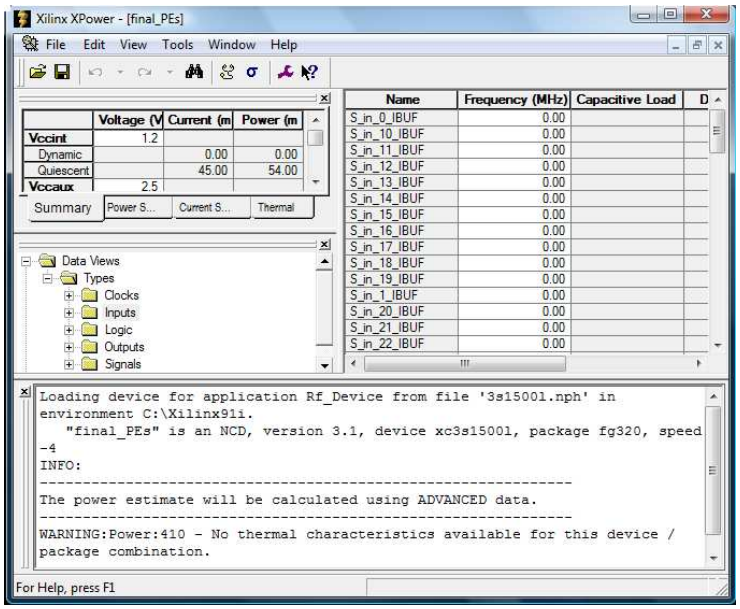


Figure B.8: ISE XPower user interface

B.3 Nonlinear Regression Analysis Tool

Nonlinear Regression (NLREG) is a powerful statistical analysis program that performs linear and nonlinear regression analysis, surface and curve fitting. NLREG determines the values of parameters for an equation, that cause the equation to best fit a set of data values. NLREG can handle linear, polynomial, exponential, logistic, periodic, and general nonlinear functions. Unlike many "nonlinear" regression programs that can only handle a limited set of function forms, NLREG can handle essentially any function whose form you can specify algebraically [116].

NLREG features a full programming language with a syntax similar to C for specifying the function that is to be fitted to the data. This allows you to compute intermediate work variables, use conditionals, and even iterate in loops. If the function being modelled is well behaved and the starting value for the parameter is not too far from the optimum value, the procedure will eventually converge to the best estimate for the parameter. This procedure is carried out simultaneously for all parameters and is, in fact, a minimisation problem in n-dimensional space, where 'n' is the number of parameters. NLREG performs true nonlinear regression analysis and curve fitting, it does not transform the function into a linear form.

The basis for the minimisation technique used by NLREG is to compute the sum of the squared residuals for one set of parameter values and then slightly alter each parameter value and recompute the sum of squared residuals to see how the parameter value change affects the sum of the squared residuals. By dividing the difference between the original and new sum of squared residual values by the amount the parameter was altered, NLREG is able to determine the approximate partial derivative with respect to the parameter. This partial derivative is used by NLREG to decide how to alter the value of the parameter for the next iteration. Fig. B.9 shows the NLREG user interface window where a power data is modelled.

B.4 FPGA Power Dissipation

Total power in an FPGA is the sum of two components: static power and dynamic power. The total power usage of an FPGA device (P_{Total}) can be broken down into total Static

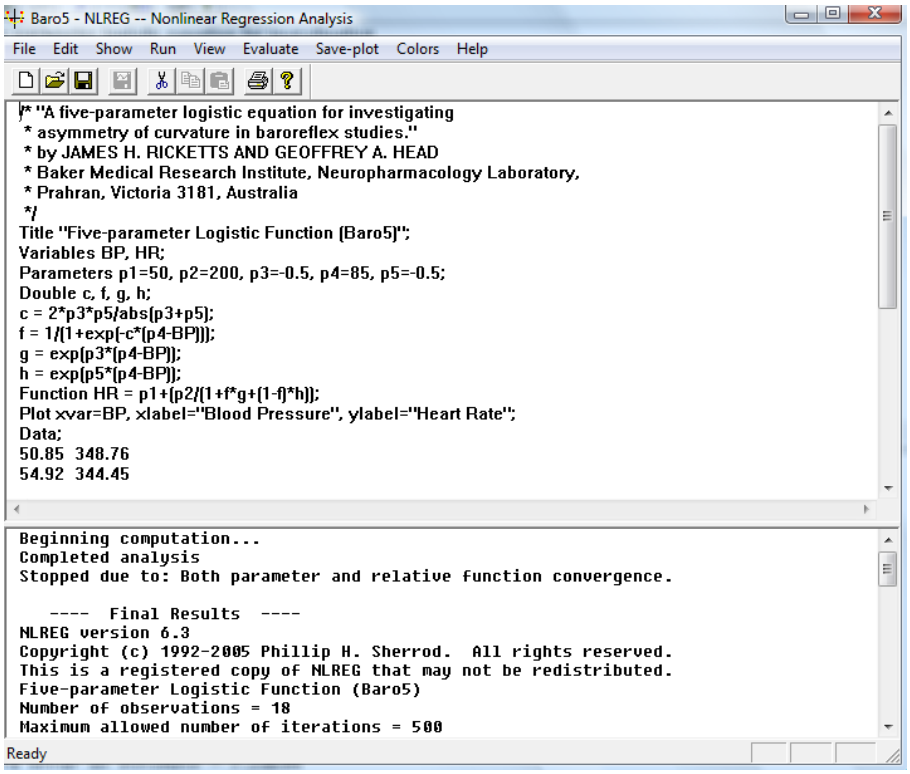


Figure B.9: NLREG user interface window

Power (SP) and total Dynamic Power (DP):

$$P_{Total} = SP + DP \tag{B.4.1}$$

B.4.1 Static Power Dissipation

Static power results primarily from transistor leakage current in the device. Leakage current is the small current that "leaks," either from source-to-drain or through the gate oxide, even when the transistor is logically "off." the device and it is, therefore, frequency dependent. The static power of an FPGA is proportional to the static current I_{dd} the current that flows regardless of gate switching (transistor is 'on' or 'off'). This is otherwise called the quiescent power. DC power dissipation can be estimated by the worst-case equivalent equation: $StP = V_{dd}I_{dd}$. Static power is inherently dependant on the architectural layout of the FPGA itself and is technology dependant. As such, it cannot be controlled by the FPGA *based* designer and will not be addressed in this work.

$$I_{CCINTQ} = I_S \longrightarrow D + I_{GATE} \tag{B.4.2}$$

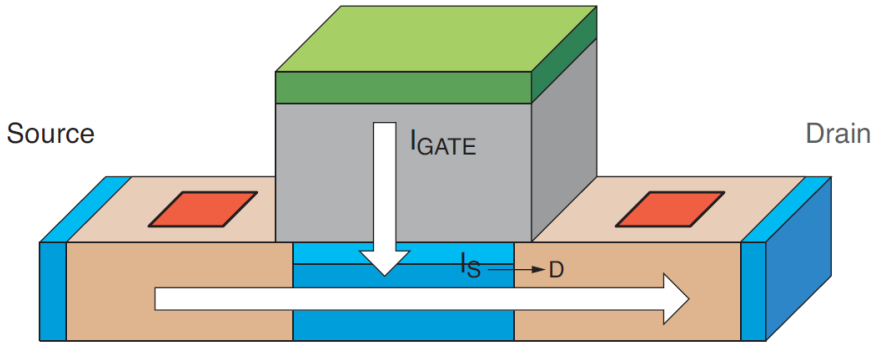


Figure B.10: Transistor leakage current for FPGA

As mentioned, static power is largely a result of transistor leakage current. Basic rules of semiconductor physics indicate that as you shrink the size of transistors (e.g., move from 90 nm to 65 nm devices), leakage current tends to increase. This predicted increase is directly related to the smaller physical dimensions of the CMOS transistors. The shorter channel lengths and thinner gate oxides that are generally used at the new process node make it easier for current to "leak," either across the channel region or through the gate oxide of the transistor. Fig. B.10 shows the two forms of transistor leakage, source-to-drain (also called sub-threshold) leakage and gate leakage.

B.4.2 Dynamic Power Dissipation

Dynamic power (DP) dissipation is caused by signal transitions in the circuit. A higher operating frequency leads to more frequent signal transitions and results in increased power dissipation. The most significant source of dynamic power consumption in CMOS circuits is the charging and discharging of capacitance. This can be modeled as

$$Power\ \ Dynamic = \sum_{i} (C_i \times f_i \times V^2) \tag{B.4.3}$$

where Ci, Vi, and fi are the capacitance, voltage swing, and operating frequency. The DP consumption of FPGAs can be separated into three main part: data-path, synchronisation and off-chip power.

Datapath power corresponds to combinational blocks and associated interconnection power. Synchronisation power is the power consumed by registers, clock lines and buffers. Data path and synchronisation power are together termed as on-chip DP. Off-chip power is the fraction dissipated in the circuit output pads. Knowledge of the relationship between

these components for a given FPGA technology is fundamental in calculating the power consumption of an FPGA-based system.

B.4.3 The Finite Radon Transform

The FRAT was first introduced in [102] as the finite analogue of integration in the continuous radon transform, with origins in the field of combinatorics. The mathematical representation of an injective form of the FRAT to ensure invertibility when applied on finite Euclidian planes has been presented in [79]. A pseudocode for the implementation of FRAT has also been provided in [79], and is reproduced in Codeblock 1. In all previous implementations of the FRAT, it has become convention to label an architecture based on the straightforward implementation FRAT pseudocode as the "*reference architecture*".

Algorithm 1 Pseudocode for the FRAT

```
1: for k = 0 : (p − 1) do
2:     n = k;
3:     for j = 0 : (p − 1) do
4:         n = n − k;
5:         if n < 0 then
6:             n = n + p;
7:         end if
8:         l = n − 1;
9:         for i = 0 : (p − 1) do
10:            l = l + 1;
11:            if l > p then
12:                l = l − p;
13:            end if
14:            FRAT(k, l) = FRAT(k, l) + f(i, j);
15:        end for
16:    end for
17: end for
18: for j = 0 : (p − 1) do
19:    for i = 0 : (p − 1) do
20:        FRAT(p, j) = FRAT(p, j) + f(i, j);
21:    end for
22: end for
```