ABSTRACT


Rapid growth in the multimedia and communications industries is fueled by the development of hardware architectures for high performance signal processing algorithms. Implementation of computationally intensive DSP algorithms has become a popular application area for digital hardware platforms such as ASICs and FPGAs. This research is directed towards improving the hardware performance of discrete wavelet transform (DWT) as a case study of DSP application that requires intensive mathematical computations. The DWT is the transformation block used in the latest multimedia compression standards such as the JPEG2000 for still image compression and H.264 for video compression.

This work presents the basis for designing an efficient implementation for a multi-level 1-D DWT hardware architecture for use in FPGAs. The proposed architecture systematically combines hardware optimization techniques to develop a flexible DWT architecture that has high performance and is suitable for portable, high speed, power-efficient devices. The hardware optimizations considered include data-interleaving for reduced FPGA resource utilization, polyphase structures for lower power dissipation and pipelining for high throughput applications. Several hardware designs with different combinations of optimizations and filter structures are designed, simulated and synthesized on the Xilinx Virtex-5 FPGA. The impact of these hardware optimization techniques on the overall DWT hardware system are analyzed and the tradeoffs between the pertinent hardware performance metrics: namely power, operating frequency, latency and resource utilization, are investigated. We illustrate that a DWT design using a pipelined and polyphase transpose direct form FIR filter coupled with data-interleaving gives the best combination of the performance metrics when compared to other DWT structures.
An Efficient FPGA Implementation of the Discrete Wavelet Transform

by
Ishita Dalal

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Electrical and Computer Engineering

Raleigh
2008

APPROVED BY:

Dr. William W. Edmonson  Dr. W. Rhett Davis

Dr. Winser E. Alexander
Chair of Advisory Committee
DEDICATION

To
My late grandfather,
Naginlal M. Shah

my grandmother,
Sarojbala N. Shah

my parents,
Uday and Trusha Dalal

and my sister,
Sonam
BIOGRAPHY

Ishita Dalal was born to Uday and Trusha Dalal on 24th June, 1984 in Mumbai, India. She received her Bachelor of Engineering (B.E) degree in Electronics from D.J. Sanghvi College of Engineering (Mumbai University) in June 2006. She joined NC State University, Raleigh in Fall 2006 to pursue graduate studies in Electrical Engineering. She has been working under the guidance of Dr. Winser Alexander as a part of the High Performance Digital Signal Processing (HiPerDSP) Laboratory at NC State University. Her area of research includes developing hardware architectures for DSP algorithms, ASIC and FPGA design and verification.
ACKNOWLEDGMENTS

My utmost gratitude goes to my advisor Dr. Winser Alexander for his expertise, unwavering support and patience throughout my research. His belief and confidence in me was the affirmation I needed to take up and succeed in this challenge. He has helped me hone my ability to conduct research, write technical documents and express my ideas. I have nothing but admiration and respect for him and working under him was truly a rewarding experience.

This journey would have been nearly impossible had it not been for my friend and mentor, Ramsey Hourani who was the torchbearer during the entire course of this research. He was always there to meet and discuss my queries, markup my chapters and give me strength when I doubted myself. A mere thank you may not suffice all the help he has given me.

I am grateful to my other committee members, Dr. Davis and Dr. Edmonson for their valuable inputs. It was a privilege to be a part of the HiPer DSP lab and I am thankful to all its members for sharing their research ideas and for the weekly presentations, especially Cranos Williams and YoungSoo Kim for helping me with \LaTeX. I would also like to thank Manav Shah for his constructive feedback, my roommates: Jyoti, Ushma and Aruna for making Raleigh a home for two years, my close friends: Sapna, Ami, Shonan, Dhvani and Akriti for their encouragement and Rishik Jariwala for proof-reading my thesis and providing me moral support during stressful times.

I am deeply indebted to my parents Trusha and Uday Dalal for their boundless love and for teaching me the true worth of education. I thank my sister Sonam for her everlasting faith in me. A very special thank you to my late grandfather Naginlal Shah and my grandmother Saroj Shah for their prayers and blessings, uncles Daxesh and Parag Shah for their priceless advice, aunts Raksha Thanawala, Bina and Roopa Shah and my cousins: Mitali, Vighnesh, Parth and Arnav for their love and warmth. Above all, I thank God for helping me achieve success in every endeavor of my life.
# TABLE OF CONTENTS

## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vii</td>
</tr>
</tbody>
</table>

## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>viii</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Background .............................. 1
1.2 Motivation .............................. 2
1.3 Contribution .............................. 4
1.4 Thesis Organization .................. 5

## 2 Background

2.1 Wavelet Theory .......................... 6
    2.1.1 Wavelet Family ....................... 8
    2.1.2 Discrete Wavelet Transform .......... 9
    2.1.3 Multiresolution analysis .......... 11
2.2 Implementation of DWT .................. 12
    2.2.1 Convolution Based DWT ............. 12
    2.2.2 Survey of DWT Architectures ....... 17
    2.2.3 Lifting based DWT ................... 21
2.3 Chapter Summary .......................... 23

## 3 Design Implementation

3.1 Floating Point Implementation ........ 27
3.2 Fixed Point Implementation .......... 29
3.3 Hardware Implementation ............... 31
    3.3.1 Hardware metrics .................. 32
    3.3.2 Filter structures .................. 35
    3.3.3 Introduction to Hardware Optimization Techniques ........ 38
3.4 Hardware Design Implementation ........ 45
    3.4.1 Basic Implementation ............... 45
    3.4.2 Polyphase Implementation .......... 47
    3.4.3 Polyphase With Pipelined Implementation ........ 49
    3.4.4 Hardware Implementation With Combined Optimizations .. 54
3.5 Chapter Summary .......................... 60
# Results and Analysis

## Bit Precision

## Virtex-5 FPGA

### ISE Project Navigator design flow

### XPower Estimator

## Results and Analysis

### Hardware Design Architectures

## Hardware Synthesis Results

### Area

### Throughput

### Latency

### Power

## Architectural Analysis

### Comparison of Architectures

### Analytical Comparison of DWT Architectures

## Chapter Summary

## Conclusion and Future Work

### Conclusion

### Future Work

# Bibliography
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>Time Frequency Localization</td>
<td>16</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>Comparison of different Filter Architectures</td>
<td>37</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Comparison of computations for polyphase and non-polyphase filters</td>
<td>40</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>Quantization Results for Fixed-Point DWT implementation</td>
<td>64</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Design Architectures</td>
<td>69</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Area Results and Percent Utilization</td>
<td>71</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Throughput Results</td>
<td>74</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>Latency Results</td>
<td>75</td>
</tr>
<tr>
<td>Table 4.6</td>
<td>Power Results for DWT architectures in design space</td>
<td>78</td>
</tr>
<tr>
<td>Table 4.7</td>
<td>Performance comparison for 16-tap, 3-level DWT Designs</td>
<td>82</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Basic Image Compression System</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Time scaling the wavelet function</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Time shifting the wavelet function</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Single DWT Block</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>3-level 8 channel DWT decomposition</td>
<td>14</td>
</tr>
<tr>
<td>2.5</td>
<td>Time-frequency graph for a 3-level 8 channel DWT decomposition</td>
<td>15</td>
</tr>
<tr>
<td>2.6</td>
<td>Coefficient-Interleaving implemented by Marino et al. [1]</td>
<td>18</td>
</tr>
<tr>
<td>2.7</td>
<td>Systolic implementation of a 3-Level DWT by Denk et al. [2]</td>
<td>20</td>
</tr>
<tr>
<td>2.8</td>
<td>Parallel implementation of the 1-D DWT by Zhang et al. [3]</td>
<td>21</td>
</tr>
<tr>
<td>2.9</td>
<td>The Lifting Scheme</td>
<td>22</td>
</tr>
<tr>
<td>3.1</td>
<td>Design Flow</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Single Precision IEEE floating point format</td>
<td>28</td>
</tr>
<tr>
<td>3.3</td>
<td>General FPGA architecture [4]</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Critical Path Delay</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Latency</td>
<td>34</td>
</tr>
<tr>
<td>3.6</td>
<td>Block Diagrams for FIR Structures</td>
<td>36</td>
</tr>
<tr>
<td>3.7</td>
<td>Filtering and downsampling</td>
<td>39</td>
</tr>
<tr>
<td>3.8</td>
<td>Example of decimation by 2</td>
<td>39</td>
</tr>
<tr>
<td>3.9</td>
<td>Polyphase structure</td>
<td>39</td>
</tr>
</tbody>
</table>
Figure 3.10 Filter zero-interleaved by factor 'k' ........................................ 42
Figure 3.11 Data-interleaving .............................................................. 44
Figure 3.12 IO waveforms for a DWT algorithm ................................. 46
Figure 3.13 Decimation by two .............................................................. 46
Figure 3.14 Polyphase implementation for a level 1 low pass filter .......... 47
Figure 3.15 Polyphase implementation for a level 1 low pass filter ........ 48
Figure 3.16 Pipelined computational cell of a Direct Form (DF) Filter ......... 50
Figure 3.17 Pipelined computational cell of a Transpose Form-II (TF-II) Filter . 52
Figure 3.18 Pipelined structures for level 1 polyphase structure \( H_e \) .... 53
Figure 3.19 3-level 8 Channel DWT structure with data-interleaving ......... 55
Figure 3.20 Our Method of implementation ........................................... 56
Figure 3.21 Pipelined computational cell of a Transpose Form-II (TF-II) Filter . 57
Figure 3.22 Details of control structure C2 ........................................... 58
Figure 3.23 Waveforms of control structure C2 ..................................... 59
Figure 3.24 Details of control demultiplexer .......................................... 59
Figure 3.25 Deinterleaving waveforms .................................................. 60

Figure 4.1 ISE Design Flow [5] ......................................................... 66
Figure 4.2 Normalized Area Graph ..................................................... 72
Figure 4.3 Toggle rates ................................................................. 76
Figure 4.4 Power Graphs ............................................................... 79

Figure 5.1 2-D decomposition of an image using the DWT [6] .................... 87
Figure 5.2 3-level 8-channel Reconstruction Filter [6] ........................ 88
Chapter 1

Introduction

1.1 Background

Video and image processing algorithms have developed at a blistering pace in the past decade and are giving rise to new innovations and applications. The new genre of multimedia applications equipped with good quality video and images makes its presence ubiquitous throughout the internet and the mobile industry. A myriad of multimedia applications, such as digital broadcasting, mobile video conferencing, video streaming, HDTV technology, is a consequence of highly sophisticated and complex signal processing algorithms used in the core of this advanced technology.

Traditional Digital Signal Processors (DSPs) have limited capabilities for processing such high volume data efficiently at real-time or near real-time rates. The trend is shifting towards the use of Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGA) to meet the increased complexity and performance requirements of these algorithms. Re-programmability of existing hardware is highly preferred due to the constantly evolving nature of multimedia technology with newer standards and applications being developed to satisfy the demand for
higher performance. Lower implementation costs and time to market are the other key factors which make FPGAs an increasingly popular choice for implementing computationally intensive digital signal processing algorithms.

Xilinx, one of the leading FPGA vendors, has introduced FPGAs with enhanced signal processing capabilities. Advanced CMOS technology, high performance logic and inherent parallelism enables FPGAs to have special multiply-accumulate (MAC) blocks within its hardware. This improves the performance of complex algorithms used for DSP applications compared to programmable DSPs. Higher throughput, lower power consumption, lower latency and efficient hardware utilization can be achieved by applying correct design optimization techniques on the hardware architecture. These goals can also be achieved by exploiting the modularity of the DSP algorithms and by making optimal use of the available resources within the FPGA. This work illustrates how appropriate hardware optimizations can be applied on computationally intensive DSP algorithms to improve their performance and efficiency for use in FPGAs.

1.2 Motivation

The integration of video, audio and data in telecommunication devices has revolutionized how the world communicates. It has proven to be useful to almost every industry: the corporate world, entertainment industry, multimedia, education and even at home. The major problems encountered with these applications are the high data rates, high bandwidth and large memory required for storage and computing resources. Even with faster internet, throughput rates and improved network infrastructure, there are major bottlenecks in transferring such high volume data through the network due to bandwidth limitations. This justifies the need to develop compression techniques in order to make the best use of available bandwidth.

In 1992, the Joint Photographic Experts Group (JPEG) released their first in-
ternational compression standard for continuous-tone still images, both in color and greyscale [7]. However, with an increase in demand for real-time applications and higher quality images, a new state-of-the-art compression standard has been developed. Almost a decade later, in 2001, the JPEG2000 compression standard with enhanced image compression techniques and new features was released. This new and improved standard is increasingly used in internet, digital photography, color facsimile, medical imaging, military video surveillance, mobile technology, multimedia and many more applications. Lossy and lossless compression, error resilience, progressive transmission of images by pixel accuracy and resolution, region of interest (ROI) coding, better image quality for the same compression ratios as that of JPEG, are some of the feature which make the JPEG2000 a preferred compression standard for these applications.

Each image compression standard undergoes the three basic operations: transformation, quantization and entropy encoding. Figure 1.1 illustrates a typical image compression system [6] with three blocks to implement the basic operations.

![Figure 1.1: Basic Image Compression System](image)

The input image is initially divided into a number of blocks or frames prior to transformation. The JPEG standard is a block-based compression standard whereas, JPEG2000 is a frame-based compression standard. The coefficients obtained after image transformation are quantized and later entropy encoded to yield the final compressed image [8]. The signal processing algorithms used in the transformation block distinguish both JPEG and JPEG2000. JPEG uses the Discrete Cosine Transform (DCT) whereas JPEG2000 uses the Discrete Wavelet Transform (DWT).
The two-dimensional DWT used in JPEG2000 is based on wavelet theory. Time-frequency localization and multiresolution analysis properties have made wavelet transforms popular in applications that require progressive transmission of images by pixel accuracy and resolution and for capturing high-frequency details [8]. The DWT algorithm is also used in other applications such as signal de-noising, speech recognition, video compression and multiresolution video.

ASICs and FPGAs are popular choices for the hardware implementations of the DWT since they can meet the high performance requirements for this algorithm. The DWT algorithm can be computationally intensive as it requires a large amount of filtering of the input signal with the wavelet coefficients along with several levels of decomposition. Over the past couple of decades a lot of research has been done on various optimization techniques for the hardware implementations of the DWT. We assess the impact of hardware optimization techniques on the performance of the convolution based DWT algorithm for this work.

1.3 Contribution

The demand for better quality images and video on mobile devices at real-time speed is driving the need to develop better digital signal processing standards for the multimedia market. Highly sophisticated signal processing algorithms with high computational requirements are used in smaller hardware platforms with high performance. FPGAs seem to be an ideal fit for image and signal processing algorithms due to their advantages of re-programmability, flexibility, faster time to market, lower non-recurring engineering (NRE) costs compared to ASICs, better performance and smaller real estate compared to DSPs.

The scope of this work is to implement and optimize a one dimensional DWT algorithm to improve the performance for use in FPGAs. The convolution based 1-D DWT is implemented at different levels of design abstraction (MATLAB and Ver-
ilog) and synthesized on an FPGA. Optimizations such as pipelining for improved throughput, decimated polyphase filter implementation for reduced power consumption and data-interleaving for reduced resource utilization are explored. The impact of these optimizations on the overall performance of the DWT designs are assessed and the results are compared. This work also develops a methodology and decision criteria such that these optimizations can be applied on other DSP algorithms that are computationally intensive and have regular data flow.

1.4 Thesis Organization

This thesis is structured in the following manner. Chapter 2 explains the theory behind the wavelet transform and discusses the concepts underlying the DWT algorithm. The hardware structure of the 3-level 8 channel DWT is studied and a literature survey of the various VLSI and FPGA implementations of the DWT is conducted. In Chapter 3, we discuss the design methodology used to implement our design and explain how the various optimization techniques affect the area, throughput, power and latency of the DWT architecture. Chapter 4 presents the results obtained by synthesizing our designs onto a FPGA and the analysis of these results is conducted. Chapter 5 concludes this work and presents the possible avenues of future work in this field.
Chapter 2

Background

A number of signal processing applications such as edge detection, feature extraction, speech recognition, echo cancelation and multimedia compression deploy the DWT. JPEG2000, the image compression engine, has adopted the DWT as its transformation standard \cite{6}. Multiresolution analysis, time-frequency localization and region of interest coding are some of the key features which makes the DWT an attractive choice. This chapter discusses the background of the basic wavelet theory, the mathematical principles underlying the DWT and the foundation for its hardware implementation. This chapter also presents different types of DWT implementations and the past works of several researchers concerning this topic.

2.1 Wavelet Theory

A wave is a continuously oscillating function of the variable being measured in time or space. According to the Fourier theory, a wave is a sum of several harmonically related sinusoids. The Fourier Transform (FT) decomposes a wave into complex
exponential functions of different frequencies as seen by Equation 2.1.

\[ x(t) = \int_{t} x(f)e^{-2j\pi ft}df \quad (2.1) \]

The Fourier transform helps determine the frequency components present in the signal being measured \( x(t) \) and has proven itself to be a very helpful tool for the analysis of time-invariant, periodic signals. However, the FT has been unsuccessful for time-variant signals such as image and speech signals whose frequency response vary with time [9].

The **Short-time Fourier Transform (STFT)** was developed to overcome the limitations of the FT and determine the frequency components present at specific times, within a time-variant signal. The STFT divides the non-stationary signal into local time sections within which the signal is assumed to be stationary. The FT is then applied to the local section of the signal with a “window” function localized in time. The STFT can be mathematically expressed as follows:

\[ STFT^{w}_{x}(t, f) = \int_{t} [x(t) \cdot \omega^{*}(t - t')] e^{-j2\pi ft}dt \quad (2.2) \]

The STFT helps determine the frequency bands present within the time-interval of the local section. Time and frequency resolution can be varied by varying the window size. For narrow windows, the time resolution is good and the frequency resolution is poor. Conversely, for wide window sizes the time resolution is poor but the frequency resolution is good [9]. Furthermore, wide windows violate the condition of stationarity assumed within the local time section. Therefore, the STFT analyzes all frequencies with uniform resolution.

In contrast, a **wavelet**, defined as a localized wave has its energy concentrated in time [6]. The wavelet transform (WT) analyzes the signal using wavelets of finite energy. The WT allows for time and frequency analysis of the signal simultaneously and studies the signal with a resolution matched to its scale. This time-frequency
localization of the WT makes it suitable to analyze time-variant signals. The remainder of the section provides a brief overview of the mathematical principles underlying the wavelet transform.

2.1.1 Wavelet Family

Wavelet functions are generated by dilations (or scaling) and translations (or shifting) of a basis function, called the “mother wavelet”, in the time (or frequency) domain [6]. Mathematically, this is expressed as:

\[\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}}\psi\left(\frac{t - b}{a}\right)\]  

(2.3)

where \(\psi_{a,b}\) denotes the mother wavelet or the basis function. The transformed signal is a function of two continuous real variables, \(a\) and \(b\), that represent the parameters for dilation and translation in the time domain respectively.

![Image](a) Mother wavelet (db8) (b) Higher scales, a>1 (c) Lower scales, 0<a<1

**Figure 2.1: Time scaling the wavelet function**

The dilation variable, \(a\), is used for time-scaling the function. Figure 2.1 shows the mother wavelet (db8 Daubechies orthogonal wavelet family) and the scaled versions of the basis wavelet function obtained by varying the dilation variable \(a\). If \(a\) is large, it causes an expansion in the time axis (high scales) and hence low frequency. This gives the “coarse” information or the larger picture of the signal. Similarly, if \(a\) is small, it gives high frequency and subsequently causes compression in time (low scales). Lower scales give the “detailed” information or the finer information of the signal.
The translation parameter $b$, is so called because it relates to the location of the wavelet function as it shifts through the signal. In Figure 2.2 we can see the position of the wavelet function at $b = 0$ and the shifted wavelet function at $b = k$. Hence, the variable $b$ is also known as the “shift” parameter in the frequency domain. This gives the time information of the signal. The term $\frac{1}{\sqrt{a}}$ in Equation 2.3 is used for normalization.

![Figure 2.2: Time shifting the wavelet function](image)

Based on the definition of wavelets given in Equation 2.3, the wavelet transform can be derived by convolving the input signal by the wavelet functions $\psi_{a,b}(t)$. Therefore, the one-dimensional wavelet transform of an input signal $x(t)$ can be expressed as:

$$W_{a,b} = \int_{-\infty}^{+\infty} \psi_{a,b}(t)x(t)dt \quad (2.4)$$

The above wavelet transform is known as Continuous Wavelet Transform (CWT) as the input signal $x(t)$ and variables $a$ and $b$ are continuous.

### 2.1.2 Discrete Wavelet Transform

In today’s “digital” world, most of the input signals are digital in nature. The CWT requires a large amount of computational time and resources due to the inte-
gration of continuous values. These computations can be reduced by using discrete values instead of continuous values and hence changing the integration to summation [6].

There is a need to transition from CWT to DWT as we transition from continuous signals to discrete signals. It is important to discretize the dilation and translation parameters, $a$ and $b$ to reduce the continuous basis set of wavelets to a discrete set of wavelets. This is done by defining the discrete wavelet parameters $a$ and $b$ as:

$$
a = a_0^m
$$
$$
b = nb_0 a_0^m \quad m, n \in \mathbb{Z} \quad (2.5)
$$

The discrete wavelet family can be represented as:

$$
\psi_{m,n}(t) = a_0^{-m/2} \psi(a_0^{-m}t - nb_0) \quad (2.6)
$$

The discretization is done by sampling the CWT using the Nyquist sampling theorem to allow perfect reconstruction of the signal during synthesis. The concept behind Dyadic sampling, a popular sampling method, is that the consecutive discrete values of $a$ and $b$ and the sampling intervals should differ by a factor of two [6]. Therefore, values $a_0 = 2$ and $b_0 = 1$ are chosen for the dyadic decomposition of the signal. We obtain the discrete wavelet family as shown by Equation 2.7 by substituting these values in the Equation 2.6.

$$
\psi_{m,n}(t) = 2^{-m/2} \psi(2^{-m}t - n) \quad (2.7)
$$

where, $\psi_{m,n}(t)$ constitutes a family of orthogonal basis functions.

For dyadic decomposition, the wavelet coefficients $c_{m,n}(x)$ can be written as:

$$
c_{m,n}(x) = 2^{-m/2} \int x(t) \psi(2^{-m}t - n) dt \quad (2.8)
$$

The wavelet series shown above is called the discrete time wavelet transform (DTWT) since the input function is continuous.
The reconstruction of the signal $x(t)$ from the discrete wavelet coefficients can be derived accordingly as:

$$x(t) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} c_{m,n}(x)\psi_{m,n}(t)$$  \hspace{1cm} (2.9)

If the input function is also discrete in nature, the transformation is called the **Discrete Wavelet Transform**.

### 2.1.3 Multiresolution analysis

In the late 1980s, Mallat proposed the multiresolution representation of signals based on wavelet decomposition [10]. Multiresolution analysis (MRA) approximates the signal $x(t)$ at different levels of resolution [6]. For a DWT, MRA is done by decomposing the input signal into different frequency bands called subbands. The input signal $x(t)$ in the Equation 2.8 above has a resolution of $2^m$. It can be decomposed into two parts of resolutions $2^{m+1}$ using MRA. One part would give the coarse approximation of the signal $(a_{m+1,n})$ and the other part would give the finer details of the signal $(c_{m+1,n})$. This can be mathematically represented by Equation 2.10.

$$x(t) = \sum_{n} a_{m+1,n}\phi_{m+1,n} + \sum_{n} c_{m+1,n}\psi_{m+1,n}$$  \hspace{1cm} (2.10)

where $\phi_{m+1,n}$ and $\psi_{m+1,n}$ are the dilation and wavelet basis function respectively.

Mallat proposed that the wavelet representation can be computed with a pyramidal algorithm (PA) based on convolutions with a quadrature mirror filter (QMF) [3]. The coefficients $a_{m,n}$ and $c_{m,n}$ can be written in terms of quadrature mirror FIR filters $G$ and $H$ as follows:

$$c_{m,n} = \sum_{k} G_{2n-k}a_{m-1,n}$$  \hspace{1cm} (2.11)

$$a_{m,n} = \sum_{k} H_{2n-k}a_{m-1,n}$$  \hspace{1cm} (2.12)
where, $G$ and $H$ are high-pass and low-pass FIR QMF respectively.

These filters are derived from the wavelet basis functions $\phi$ and $\psi$ and hence satisfy the perfect reconstruction criterion for QMF. Ingrid Daubechies derived compactly supported orthonormal wavelets called the Daubechies wavelet family [11], which are suitable for discrete wavelet analysis and hence used in this thesis.

### 2.2 Implementation of DWT

The DWT can be implemented using general purpose processors (GPP) or hardware platforms such as ASICs or FPGAs. Software implementations of the DWT are very flexible but for large input data, these implementations may fail to meet the stringent timing constraints required for real time applications. This is due to the sequential nature of the algorithm that is implemented on the GPP. Hardware implementations of the DWT, though not very flexible, are preferred over software implementations due to their inherent parallelism and better performance for real time applications.

The DWT can be traditionally implemented in hardware using two popular methods: Convolution based DWT and the Lifting based DWT. While both the implementation methods have been described, this work focuses on the convolution based DWT. A review of the optimizations adopted by various researchers to improve the performance of the algorithm and the efficiency of the convolution based DWT hardware is also provided.

#### 2.2.1 Convolution Based DWT

Mallat’s pyramid algorithm computes the one dimensional (1-D) convolution based DWT at different levels of resolution. The first level decomposition can be represented by using the block diagram illustrated in Figure 2.3:
The input sequence $x(n)$ in Figure 2.3 is convolved with the quadrature mirror filters $H(z)$ and $G(z)$ and the outputs obtained at each level are decimated by a factor of two. After downsampling, alternate samples of the output sequence from the low pass filter and high pass filter are dropped. This reduces the time resolution by half and conversely doubles the frequency resolution by two. The computation for the output sequences $y_L$ and $y_H$ from the low pass filter and high pass filter can be represented with the following equations:

$$y_L(n) = \sum_{i=0}^{n-1} H(i)x(2n - i)$$
$$y_H(n) = \sum_{i=0}^{n-1} G(i)x(2n - i)$$

(2.13)

The analysis filter stage recursively decomposes the input signal into the approximation (low frequency) signal and the detail (high frequency) signal at the next lower resolution. For an input sequence of $N$ samples in length, the wavelet tree can be extended to $J = \log_2 N$ levels of decomposition. One basic 3-level 8-channel 1-D DWT architecture is illustrated in Figure 2.4.
Figure 2.4: 3-level 8 channel DWT decomposition
The hardware implementation of the DWT architecture as shown in Figure 2.4 is realized by a number of cascaded filter blocks followed by scaling. The above diagram can be better understood with the aid of the time-frequency graph as shown in Figure 2.5.

Figure 2.5: Time-frequency graph for a 3-level 8 channel DWT decomposition

Point A in Figure 2.4 is the input to level 1. Point B is the output samples of level 1 and input samples of level 2. Similarly, point C is the output samples of level 3 and point D is the output samples of level 3. The dots in Figure 2.5 represent the position of the input and output sequences corresponding to points A, B, C and D, on the time scale plane. For example, at point A, the input sequence $x(n)$ is being filtered at every clock cycle. The clock frequency bandwidth for input $x(n)$ is $F_{Hz}$ and the number of samples is $N$. The frequencies spanned by the input sequence is 0 - $F$ Hz. Table [2.1] illustrates the frequencies spanned, frequency bandwidth and the
number of samples corresponding to the output sequences from levels one, two and three. The “Point” in Table 2.1 corresponds to the A, B, C and D points in Figure 2.4 and Figure 2.5.

Table 2.1: Time Frequency Localization

<table>
<thead>
<tr>
<th>Point</th>
<th>Sequences</th>
<th>Frequency Range</th>
<th>Frequency Bandwidth</th>
<th>Number of Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$x(n)$</td>
<td>0 - $F$ Hz</td>
<td>$F$ Hz</td>
<td>N</td>
</tr>
<tr>
<td>B</td>
<td>$y_L(n)$</td>
<td>0 - $F/2$ Hz</td>
<td>$F/2$ Hz</td>
<td>N/2</td>
</tr>
<tr>
<td></td>
<td>$y_H(n)$</td>
<td>$F/2$ - $F$ Hz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>$y_{LL}(n)$</td>
<td>0 - $F/4$ Hz</td>
<td>$F/4$ Hz</td>
<td>N/4</td>
</tr>
<tr>
<td></td>
<td>$y_{LH}(n)$</td>
<td>$F/4$ - $F/2$ Hz</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$y_{HL}(n)$</td>
<td>$F/2$ - $3F/4$ Hz</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$y_{HH}(n)$</td>
<td>$3F/4$ - $F$ Hz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>$y_{LLL}(n)$</td>
<td>0 - $F/8$ Hz</td>
<td>$F/8$ Hz</td>
<td>N/8</td>
</tr>
<tr>
<td></td>
<td>$y_{LLH}(n)$</td>
<td>$F/8$ - $F/4$ Hz</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$y_{LHL}(n)$</td>
<td>$F/4$ - $3F/8$ Hz</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$y_{LLH}(n)$</td>
<td>$3F/8$ - $F/2$ Hz</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$y_{HLL}(n)$</td>
<td>$F/2$ - $5F/8$ Hz</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$y_{HLH}(n)$</td>
<td>$5F/8$ - $3F/4$ Hz</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$y_{HHL}(n)$</td>
<td>$3F/4$ - $7F/8$ Hz</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$y_{HHH}(n)$</td>
<td>$7F/8$ - $F$ Hz</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Therefore from Table 2.1 and Figure 2.5 we conclude that after every level, the filtering and subsampling results in half the number of samples and half the frequency bands spanned. This procedure of dividing the input signal into the different frequency bands is also known as the subband decomposition.

FIR filter blocks are important building blocks of these DWT architectures. Many DSP algorithms use the FIR filters due to their inherent stability, linearity and causality. These FIR filters are composed of multipliers, adders and delay units. Multipliers within the FIR filters consume a large hardware area. The layout area for an ASIC implementation of the DWT can be reduced by removing the redundant filters in this architecture, without sacrificing much on the performance. Therefore, this architecture, however easy to implement, has a large area and may have large critical path delay and latency depending on the filter length and order. Extensive research has been conducted in the wavelet hardware domain to address these issues and designers have suggested various optimization techniques to improve the hardware performance of the DWT [12].

### 2.2.2 Survey of DWT Architectures

Baganne et al. [13] implemented Mallat’s 3-level PA architecture directly but used only four outputs, namely $y_H$, $y_{LH}$, $y_{LLH}$ and $y_{LLL}$, and hence only three cascaded DWT blocks. They also used single line input to help share the delay blocks used by the direct form (DF) lowpass (LP) and highpass (HP) filters within each DWT block. The architecture was modular, had low design complexity, low hardware latency and could be easily expanded to further levels of decomposition. However, this architecture had a large critical path delay and downsampling by two after every stage led to increased power dissipation due to performing unnecessary computations. These issues can be addressed by applying appropriate hardware pipelining techniques within the DF filter and by using polyphase filters instead of decimating after filtering [14].

Polyphase and pipelining techniques for a 1-D DWT convolution based archi-
architecture were implemented by Marino et al. for a low power and high speed VLSI architecture [1]. Their architecture used transpose form (TF) filters to reduce the critical path delay to one multiplier and one adder. Delay blocks were placed at the outputs of each level to ensure lower critical path delays with minimal increase in hardware latency. Furthermore, the input signal was divided into even and odd parts and filtered using polyphase filters instead of downsampling after filtering. Using polyphase techniques ensured low power by computing only the values that have to be saved. For the second level of decomposition and higher, this architecture used circular shift registers (CSR) to interleave the coefficients of the LP and HP filters. The coefficients in the CSRs for level 2 is 2, for level 3 is 4, for level 4 is 8 and so on. Figure 2.6 illustrates how coefficient-interleaving was implemented for levels 2 and 3.

![Coefficient-Interleaving implemented by Marino et al. [1]](image-url)

Figure 2.6: Coefficient-Interleaving implemented by Marino et al. [1]

In Figure 2.6, the $h$ and $g$ are the low pass and high pass filter coefficients used in the CSRs. The purpose of coefficient-interleaving in Marino’s work was to reduce the number of multipliers within the design. However, the use of CSRs for coefficient-interleaving may increase the dynamic power required for the switching between the coefficients.

Additionally, for levels three and higher, the architecture in [1] did not use polyphase
techniques which lead to wastage of computations. Additionally, the critical path delay for levels higher than two was large due to lack of pipelining and feedback loops to the multiplexers. Each stage differs from the previous stage which made this design complex and not easily expandable to further levels of decomposition. The use of TF filters in this architecture might exhibit broadcasting at the input depending on the filter order. Broadcasting occurs when there is a direct wire feeding a large number of combinational blocks such as multipliers. Broadcasting requires an increased voltage at the input to drive the signal through all the multipliers. This has a tendency to increase the power dissipation [15]. However, this work is an example of how optimization techniques such as polyphase, pipelining and interleaving can be applied to the architecture and FIR filters to make them application specific and improve the performance in computationally intensive DSP algorithms.

A similar technique for coefficient-interleaving was presented by Shahid Masud and John V.McCanny [16] to implement a three level 1-D orthonormal DWT on a Xilinx 4052XL FPGA. The first stage implemented the coefficient-interleaved DF FIR filters. Polyphase techniques were not used for this architecture. However, there was no wastage of computations because the coefficients in the CSRs were alternated depending upon the even and odd parts of the signal. This implementation made use of buffer memory to temporarily store intermediate output values. Each stage differed from the previous stage making the design complex. Also, for higher stages of implementation, this architecture required significant switching activity to correctly synchronize the data and coefficients which could increase the power required for the design. Additionally, the critical path increased with each stage and the throughput was reduced by two after level 2 and by four after level 3 which may make this architecture unsuitable for real time applications.

Denk et al. employed a single processor to compute a three-level 1-D DWT [2] as illustrated in Figure 2.7. The single processor consisted of many processing units (PU) arranged in a systolic manner. This architecture implemented the folding technique based on the recursive pyramidal algorithm (RPA) [17], wherein the outputs of the
successive levels of decomposition were interleaved with the original input in order to use only one processor. The advantage of such a fold-back design was that it utilized less hardware area. However, architectures that employ the folding technique require a fairly complex controller to adjust the data flow from one decomposition level to the next. Additionally, the critical path delay could be high for higher filter orders such as the ones considered in our work. The critical path delay could be reduced by appropriately pipelining the design which in turn would increase the latency. Similar folded architectures which used only a single processing element and implemented the RPA were suggested by Premkumar et al. [18].

![Figure 2.7: Systolic implementation of a 3-Level DWT by Denk et al. [2]](image)

Zhang et al. [3] proposed an architecture that required two PUs and one buffer to implement the convolution based DWT using folding techniques. This parallel implementation of the 1-D DWT was based on the modified RPA (MRPA) suggested by Chakrabarti et al [19]. Figure 2.8 illustrates the parallel implementation proposed by Zhang et al.

Figure 2.8 shows the use of two PUs and one buffer. The concept behind the use of two PUs was that the number of computations performed by the level 1 is greater than or equal to the sum of the computations done by remaining levels. Hence, while the PU1 computes the outputs from the first level, the PU2 simultaneously computes the outputs of the other levels. The buffer was used to temporarily store
the values of the levels that were used in computing the outputs at higher levels of decomposition. A multiplexer was used to correctly interleave the outputs obtained from PU1 and folded values of PU2 into the buffer. The outputs obtained from the buffer were used as the input to the PU2. Since this architecture used two PUs, it is an improvement over the architecture that uses a single processing element [2] [18] in terms of latency. Uzun and Amira [20] implemented a similar architecture to implement a discrete biorthogonal wavelet transform on a Xilinx Virtex 2000-E FPGA. Marino et al. [1] had also implemented another 1-D DWT architecture, which used the folding technique that required two PUs to increase the efficiency of the design. These folded architectures are efficient in terms of area for three-level four channel or 5 channel DWT architectures but not for a three level eight channel implementation since the number of PUs would increase which would lead to an increase in the hardware area.

2.2.3 Lifting based DWT

The lifting scheme is a relatively new method to implement the discrete wavelet transform. It was originally introduced by W. Sweldens to implement second generation wavelets [21]. However, the first generation wavelet family such as the Daubechies wavelet family, can be implemented by the lifting scheme using the ladder structure [22].
The lifting scheme is implemented using three steps: splitting, predict and update step. Figure 2.9 illustrates how the lifting scheme can be implemented using these steps. The diagram shows the lifting scheme for a LeGall(5,3) biorthogonal wavelet family which requires only one predict and one update step. The predict and update computational cells are modular and can be replicated for higher order filters such as the biorthogonal (9,7) filters used by JPEG2000 [6].

![Diagram of the Lifting Scheme](image)

**Figure 2.9: The Lifting Scheme**

The **splitting step** divides the input signal $x$ into the even signal $x_e$ and the odd signal $x_o$ using the Lazy Transform [21]. The lifting scheme assumes that the two sets are closely correlated hence given any one set one can predict the other set. Therefore, the splitting step also takes polyphase decomposition into consideration.

The **predict step** computes the odd sample by adding the predicted values of the even samples $P(x_e)$ with the detail signal $d$.

$$x_o = P(x_e) + d$$ (2.14)

The detail or difference signal $d$, is the difference between actual odd values and the predicted values.

$$d = x_o - P(x_e)$$ (2.15)
The predicted value of the odd sample is the average of the two even neighboring samples. In case of linear values, the detail signal is very small due to high correlation between the even and odd samples. For ideal cases, the detail coefficient should be zero.

The update step updates the approximation signal $s$ using the detail signal $d$ obtained from the previous predict step.

$$s = x_e + U(d)$$  \hspace{1cm} (2.16)

The even coefficients can be calculated as follows:

$$x_e = s - U(d)$$  \hspace{1cm} (2.17)

The final outputs $d$ and $s$ are the detail and the approximation signal of $x$, respectively. For a 3-level 8-channel lifting based DWT, we could cascade the computational cell, as the one shown in Figure 2.9, to form the binary tree structure or we could apply the folding technique on the lifting scheme. Optimizations techniques applied on the convolutional based DWT can also be extended to the lifting based architecture due to their regular data flow. However, for this work we consider only the convolution based DWT architecture and establish a foundation for combining multiple optimizations into a single system.

### 2.3 Chapter Summary

In this chapter, we discussed the mathematical principles of the wavelet theory and theoretical foundation of the discrete wavelet transform. We also discussed the convolution based and the lifting based approaches used for the hardware implementation of the DWT. We further conducted a state-of-the-art literature survey of the convolutional based approach and the optimizations used by other designers to improve the hardware performance of the DWT architecture. While many of the convolution based DWT architectures implemented in the literature use the folding technique to
reduce the area, we will implement an architecture that will reduce the area without sacrificing the throughput and the latency of the system. The next chapter, discusses in detail the various optimizations techniques used to improve the efficiency of the convolution based 3-level 8-channel DWT architecture. We also describe how we implemented the various optimizations on the basic DWT and how these optimization impact the throughput, latency, power and the area requirements of the hardware.
Chapter 3

Design Implementation

In this chapter, we present the design and implementation of different hardware architectures for a convolution based 3-level, 8-channel 1-D DWT decomposition block. These architectural blocks are synthesized and mapped onto a Xilinx Virtex-5 FPGA. This work also investigates the differences in performance of DWT architectures when appropriate hardware optimizations are applied to them. These optimization techniques assist us in developing an efficient architecture to implement a 3-level 8 channel DWT. We use the design flow illustrated in Figure 3.1 in order to accomplish our goal.

1. Model the initial algorithm in MATLAB using floating point computations. This model uses the same 1-D DWT architecture that was used for the basic hardware implementation suggested by Mallat [10].

2. Refine the floating point model to a behavioral fixed point model to achieve bitwise accuracy (using MATLAB). This fixed point model can be used to generate test sequences to validate the hardware. The fixed point model serves as a reference for the hardware and generates the same output results as expected from the hardware implementation.

3. Capture the behavioral algorithm using a register transfer language (RTL) such
Figure 3.1: Design Flow

as Verilog to obtain a functionally correct hardware implementation of the 1-D DWT. This model uses two’s complement fixed point arithmetic. The performance analysis framework (PAF) [23] assists us in deciding the different filter structures. The results obtained from the RTL design of the DWT architecture are simulated (using Xilinx ISE software) and compared with the results obtained from the fixed point model to verify the correctness of the hardware
design.

4. Apply optimization techniques to improve the hardware efficiency. The initial DWT hardware model is modified to include optimizations that improve throughput, power consumption, latency and hardware resource utilization.

5. Analyze performances of the hardware architectures in the design space and determine an efficient architecture for the FPGA implementation. This is an incremental design and analysis process.

The tools required for the completion of this work are MATLAB and Xilinx ISE software. MATLAB (from The Mathworks) is a high-level of technical computing language for algorithm development [24]. It has built-in functions that can help solve technical problems more conveniently than with C or C++. MATLAB has the DWT toolbox which has the functions to generate wavelet coefficients and to compute the DWT which assists us in developing our initial 3-level DWT algorithm faster. Xilinx ISE allows us to implement our architecture, simulate and synthesize our design on the FPGA used for our work. The cycle-accurate RTL models are coded using Verilog, which is the popularly used HDL for digital systems.

3.1 Floating Point Implementation

For most digital hardware systems, the coefficients, inputs, intermediate outputs and the final outputs are stored in finite word registers using the binary number format. This requires quantization of floating point values depending on the number of bits used for precision. The representation of the IEEE single precision floating point format [25] is shown in Figure 3.2.
Figure 3.2: Single Precision IEEE floating point format

where $s$ is the sign bit, $e$ is the biased exponent and $m$ is an unsigned (normalized) mantissa. The actual exponent can be determined by $[e - bias]$ where $bias = (2^{E-1} - 1)$. For the IEEE single precision format, $E$ is 8 bits, hence, $bias$ is 127. A floating point number can be represented using Equation 3.1 [4].

$$x = (-1)^s 1.m 2^{e-bias} \tag{3.1}$$

The roundoff and overflow errors associated with quantization affect the mantissa of the floating point number. For accurate computations with floating point arithmetic, the exponents of the operands might have to be modified. For example, for addition operations, the exponents of the two operands should have the same value. If the exponents are not equal, they have to be adjusted such that they are lined up. Similarly, for multiplication, the mantissas are multiplied and the exponents are added [26]. Therefore, the use of floating point arithmetic to implement hardware systems requires complex hardware, which slows down the speed of the processor and increases power dissipation [4].

Floating point implementation proves very useful in helping us conceptualize our design. It also provides a solid base for comparing the results of the fixed point and Verilog implementations. Hence, it is worthwhile to develop the floating point implementation using a high level language such as MATLAB. Lower levels of design require fixed point implementations to fully refine the DWT algorithm to synthesizable hardware.
We begin the development of the floating point three-level 1-D DWT algorithm in MATLAB by developing a basic DWT block as shown in Figure 2.3. The inputs to the basic DWT block function are: the input signal, the low pass filter coefficients and the high pass filter coefficients. The input signal is convolved with the low pass and the high pass filter coefficients using the built-in MATLAB function `conv`. The outputs obtained after filtering are downsampled by removing every other odd sample.

The alternate built-in MATLAB function to perform the computation for a single level DWT block decomposition is `dwt` [24]. The filter coefficients are obtained from the Daubechies orthogonal family. For this thesis, we use the 'db8' filter family for its smooth spectral properties which make them suitable for many speech and image processing algorithms [11]. Additionally, the db8 family filters are 16 tap filters which are an appropriate choice to illustrate the complexity of the DWT algorithm for higher order filters. The hardware implementations considered in this work are applicable to other wavelet families and various filter orders. The decomposition and reconstruction filter coefficients for the db8 wavelet family can be obtained from the MATLAB function `wfilters`.

A three level implementation of a 1-D DWT can be done by cascading the DWT blocks using appropriate input signals as shown in Figure 2.4. Floating point computations have the advantages of a large dynamic range, high resolution and computational precision, but the disadvantages of slow speed and high system complexity.

### 3.2 Fixed Point Implementation

Fixed point numbers have a predetermined exponent as opposed to the varying exponent for floating point numbers. Additional logic is not required to compute the exponents which significantly speeds up the computation process. The system hardware required for fixed point implementations is less complex which also reduces power dissipation. Hence, in spite of accuracy and computational precision of floating
point computations, many hardware platforms, such as the ASICs and FPGAs prefer the use of fixed point arithmetic.

The dynamic range for $N$ bit fixed point number is $[-2^{N-1}, 2^{N-1} - 1]$. The $Q_{m,n}$ format is used to represent $N$ bit fixed point numbers where the MSB bit is the sign bit, $m$ is the number of bits used to represent whole numbers and $n = N - (m + 1)$ is the number of bits used to represent fractional bits [26]. We use the $Q_{0,n}$ format to model the inputs, coefficient and outputs using fixed point numbers for our work. This representation assumes that the floating point inputs and coefficients are normalized in the range of (-1,1). Therefore, the range of $N$ bit fixed point numbers in $Q_{0,n}$ format is $[-(1 - 2^{-n}), 1 - 2^{-n}]$, where $n = N - 1$.

The twos complement representation is widely used to represent fixed point numbers in hardware systems. A value of $2^N$ can be added to negative numbers to obtain their corresponding twos complement numbers. Hence, instead of negative numbers having the range of $[-2^{N-1}, 0)$, the range for twos complement negative numbers is $[2^N - 2^{N-1}, 2^N)$. The use of twos complement format to represent numbers makes arithmetic computations relatively straightforward since it eliminates the need for a sign bit [26].

We proceed with our design cycle by implementing the DWT algorithm using fixed point numbers. Scaling techniques need to be applied on floating point numbers, to obtain $N$ bit fixed point integers. The floating point input sequence is scaled using Equation 3.2 to represent the fixed point numbers within the specified dynamic range.

$$InputSF = \frac{2^{N-1} - 1}{\max(|x|)}$$

$$x_{scaled} = \lfloor (InputSF \times x) \rfloor$$

(3.2)

Where $x$ is a range of input and $InputSF$ is the factor by which the inputs are scaled. Every floating point input sample is multiplied by the computed $InputSF$. The re-
sult obtained is floored or truncated to obtain its corresponding \( N \) bit fixed input sequence \( x_{\text{scaled}} \). The other alternative quantization modes to \textit{floor} in MATLAB are rounding, ceil and fix. Scaling using these quantization modes give different results, but the \textit{round} mode gives most accurate results when compared with floating point values. However, the round mode requires more complex hardware. The floor mode is found to be the simplest quantization method to implement in hardware and is, therefore, commonly used for fixed-point representation [27].

Operations such as multiplications and additions in the filtering operation may lead to overflow of bits. For hardware systems, the intermediate outputs that result out of these operations also require scaling or truncation. To avoid overflow caused due to finite number of registers, the filter coefficients are scaled by using the output overflow criterion [15] [26]. The filter coefficients \( b(k) \) are scaled using Equation 3.3.

\[
\begin{align*}
\text{FilterSF} &= \frac{y_{\text{max}}}{x_{\text{max}} \cdot b_{\text{sum}}} \\
b_{\text{scaled}} &= \lfloor (\text{FilterSF} \cdot b) \rfloor 
\end{align*}
\]

Where, \( y_{\text{max}} \) is the maximum value the output can take depending on the size of output registers, \( x_{\text{max}} \) is the maximum value of the input and \( b_{\text{sum}} \) is the sum of the absolute values of all the filter coefficients \( b(k) \). The value, \( \text{FilterSF} \) is the factor by which the filter coefficients are scaled such that there is no overflow and \( b_{\text{scaled}} \) are the scaled filter coefficients. Therefore, the result of filtering a \( N \) bit input sample with \( N \) bit filter coefficients is a \( 2N \) bit output sample.

### 3.3 Hardware Implementation

The next step in the design process is to capture a functionally correct hardware algorithm using Verilog RTL. The design is verified by comparing the results obtained from the RTL design with the results obtained from the fixed point implementation.
The waveforms of the MATLAB and Verilog models are plotted to verify functional correctness.

The basic binary tree structure implementation for a 3-level, 8-channel convolution based architecture is described in Section 2.2. The architecture implemented in Figure 2.4 has several design inefficiencies. These include large critical path delay for cascaded FIR filters, poor hardware utilization due to the use of redundant filters for level 2 and higher and inefficient power dissipation due to the wastage of every alternate computation performed at the output of the decimation filters. We address these inefficiencies through an incremental design and analysis process to improve performance. We realize our final architecture by applying a combination of various optimizations to the basic design to achieve good hardware utilization, reduced critical path and efficient power utilization. The following hardware performance metrics assist in analyzing the performance improvement of the hardware architecture.

### 3.3.1 Hardware metrics

The performance of the architecture is evaluated with respect to area, throughput, latency and power consumption. We define the hardware performance metrics used in our work as listed below.

1. **Area**

   The traditional FPGA structure consists of three key elements: a two-dimensional array of logic elements, input output blocks (IOBs) and programmable interconnects (PIs). The logic elements in Xilinx FPGAs are called configurable logic blocks (CLB). These CLBs are made up of look-up tables (LUT) and flip-flops and can implement both, combinational and sequential logic. The newer generation of FPGAs, such as the one used in our work have RAM blocks and embedded DSP blocks for multiply and accumulate (MAC) intensive DSP algorithms. Figure 3.3 gives us an idea of how the resources in an FPGA are distributed [28].
The DSP blocks within the FPGAs are extremely effective for digital signal processing applications and can achieve frequencies of up to 550MHz [5]. The number of logic cells, DSP blocks, CLBs, IOBs and RAM within the FPGA vary with different FPGA families and their architecture is vendor-specific. However, the FPGA resources are limited and should be prudently used. In order to achieve the design objective of minimum area, we focus on reducing the number of resources consumed by mathematically intensive algorithms on an FPGA. This enables large MAC intensive designs to fit within the chosen FPGA.

2. **Throughput**

Critical path delay is the time taken by a signal to traverse through the maximum feasible combinational path between two registers in the design [29]. This is illustrated in Figure 3.4.
The time taken for a signal to pass through the critical path determines the operational clock frequency and hence the **throughput** of the system. We define throughput as the number of samples obtained in one second, for our work.

3. **Hardware Latency**

**Latency** is the time taken by a signal to pass through a system. It is the time difference when the input is applied and when the corresponding output is obtained. Figure 3.5 illustrates an example of the latency. In Figure 3.5 we observe that the input signal at $t = 3$ is delayed by $k = 2$ clock cycles, and
the corresponding output signal is obtained at the output at \( t = 5 \). Hence the latency is 2 clock cycles.

4. Power

The power consumption of an FPGA is a critical design constraint for small hardware platforms such as portable electronics which have limited battery life [4]. We analyze power based on the number of computations within the system and the dynamic power dissipated during the switching associated with these computations.

3.3.2 Filter structures

The SFG of the basic architecture as shown in Figure 2.4 can be implemented by using identical filter structures for the low-pass and high-pass signal decomposition. The transfer function for a FIR filter system in the \( z \) domain is shown in Equation 3.4.

\[
H(z) = \sum_{k=0}^{M-1} b_k z^{-k} \tag{3.4}
\]

where \( H(z) \) is the output sequence \( Y(z) \) and \( b(k) \) and \( M \) in Equation 3.4 are the filter coefficients and the filter length respectively.

This transfer function has a variety of realization structures. The FIR filters used in the DWT architecture can be implemented by using any of the conventional filter structures shown in Figure 3.6. The block diagrams in Figure 3.6 are cascaded realizations of an FIR filter of order \( M = 5 \). We observe that the filter blocks are modular in nature with regular data flow and can be easily extended to any order filter by simply cascading the computational cell. The input and output cells of the
filter may vary depending on the choice of filter structures.

The first step towards the hardware implementation of the DWT algorithm was to choose the type of FIR filter block. The filter blocks generated by the Performance
Table 3.1: Comparison of different Filter Architectures

<table>
<thead>
<tr>
<th>Filter Type</th>
<th>Critical Path</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF</td>
<td>$T_{\text{mult}} + (M + 1)T_{\text{add}}$</td>
<td>0</td>
<td>$(M + 1)A_{\text{mult}} + (M + 1)A_{\text{add}} + M A_{\text{reg}N}$</td>
</tr>
<tr>
<td>TF</td>
<td>$T_{\text{mult}} + T_{\text{add}}$</td>
<td>$M$</td>
<td>$(M + 1)A_{\text{mult}} + (M + 1)A_{\text{add}} + M A_{\text{reg}2N}$</td>
</tr>
<tr>
<td>DF-II</td>
<td>$T_{\text{mult}} + T_{\text{add}}$</td>
<td>0</td>
<td>$(M + 1)A_{\text{mult}} + (M + 1)A_{\text{add}} + M A_{\text{reg}2N}$</td>
</tr>
<tr>
<td>TF-II</td>
<td>$T_{\text{mult}} + (M + 1)T_{\text{add}}$</td>
<td>0</td>
<td>$(M + 1)A_{\text{mult}} + (M + 1)A_{\text{add}} + M A_{\text{reg}N}$</td>
</tr>
</tbody>
</table>

Analysis Framework (PAF) [23] provided a starting point for the implementation of the DWT. It gave an estimate of the latency, throughput and the area utilized by the filter which assisted us in the selection and analysis process. The choices for cascaded filter blocks generated by the framework are: direct form (DF) filter, transpose (TF) form filter, direct form-II (DF-II) and transpose form-II (TF-II).

Table 3.1 gives an approximate idea of the critical path delay, latency and the area associated with $M$ order filter structures. There is some latency inherent in FIR filters due to the design of the filter coefficients, however we have not included that in Table 3.1. We assume $T_{\text{mult}}, T_{\text{add}}$ and $T_{\text{reg}}$ are the times in ns required by a signal to pass through the multiplier, adder and the delay unit respectively. $A_{\text{mult}}$ and $A_{\text{add}}$ are the area occupied by the multiplier and the adder respectively. We use the notations $\text{reg}N$ and $\text{reg}2N$ to represent the registers occupied by $N$ bits and $2N$ bits, respectively.
The TF and DF-II filters have low critical path delays but have the problem of data-broadcasting. DF filters have low latency but a large critical path delay. TF-II filters also have low latency but large critical path delay. For our work, we use the DF and TF-II structures. We choose DF filters to get an assessment of the performance improvement over Baganne’s architecture [13]. We have implemented the TF-II filter to illustrate that the choice of filters can affect the performance of the hardware. The PAF gives us the flexibility to implement different filter architectures with ease.

3.3.3 Introduction to Hardware Optimization Techniques

Optimizations are based on the technology on which the design is synthesized and the tools that can help improve the hardware utilization and reduce the power dissipated. However, optimizations at the architectural level can significantly impact the hardware performance of the algorithm. This section lists the commonly used hardware optimization techniques applied to DSP hardware algorithms.

Polyphase

Multirate DSP systems are commonly used in many practical signal processing applications that require different sampling rates. Downsampling and Upsampling are common techniques used for sampling rate conversion in analysis and synthesis filters such as the QMF banks [30]. The input is filtered and downsampled by a factor of two at every stage of the DWT algorithm. Downsampling, also known as decimation, can be easily implemented as shown in Figure 3.7.

where \( H(z) \) is the filter transfer function, \( x(n) \) is the input signal, \( y'(n) \) is the filtered output and \( y(n) \) is the final decimated output signal. The input clock frequency is \( F \) Hz and the output clock frequency is \( F/2 \) Hz. Figure 3.7 illustrates a sample waveform for the output sequence after filtering \( y'(n) \) and the output sequence after decimation \( y(n) \).

Decimation by two discards every alternate sample computed by the filter. This
wastage of computations can be avoided by using two-phase polyphase filters as shown in Figure 3.9. The polyphase technique is widely used in multi-rate high speed filters which involve different sampling rates for the input and output signals [14]. As an example, the transfer function for a 6-tap FIR filter is
Table 3.2: Comparison of computations for polyphase and non-polyphase filters

<table>
<thead>
<tr>
<th>Filter</th>
<th>Input</th>
<th>Multiplications</th>
<th>Additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filtering followed by downsampling</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$H_e$</td>
<td>8</td>
<td>50</td>
<td>400</td>
</tr>
<tr>
<td>$H_o$</td>
<td>8</td>
<td>50</td>
<td>400</td>
</tr>
<tr>
<td>Total Polyphase computations</td>
<td></td>
<td>800</td>
<td>700</td>
</tr>
</tbody>
</table>

\[ H(z) = b(0)z^0 + b(1)z^{-1} + b(2)z^{-2} + b(3)z^{-3} + b(4)z^{-4} + b(5)z^{-5} \] (3.5)

The above filter can be split into its corresponding polyphase filters as shown below:

\[ H_{even}(z) = b(0) + b(2)z^{-2} + b(4)z^{-4} \]
\[ H_{odd}(z) = z^{-1}(b(1) + b(3)z^{-2} + b(5)z^{-4}) \]

where

\[ H(z) = H_{even} + z^{-1}H_{odd} \] (3.6)

The advantage of this method is that the same functionality can be achieved by computing only the necessary values. This avoids wastage of computations which in turn reduces the power dissipated. In case of DF filters, the critical path through the filter is also reduced since the filter is divided into two filters for decimation by 2. In general, for decimation by factor $D$, the filter can be split into $D$ polyphase filters.
Table 3.2 shows the number of multiplications for polyphase and non-polyphase structures. The table shows there is a saving of half the multiplications and more than half of the additions. We confirm that the use of polyphase technique for down-sampling filters allows calculation of only the values to be saved which reduces the number of computations. Hence, the power dissipated by decimation filters that use the polyphase technique is less than that for those that do not use polyphase implementation.

**Pipelining**

The pipelining technique is used in hardware systems for concurrent processing. Concurrent processing of data divides the computational load between multiple processing elements which in turn helps achieve high processing rates for large designs. Pipelining techniques can reduce the critical path delay of the system and can eliminate broadcasting and global interconnections within the design. There are two techniques used to pipeline the design: the interleaving technique and the delay technique [31]. The techniques developed by Dabbagh et al. are based on the properties of the $z$-transform and can be applied to modular structures such as the digital FIR filter used in our design. We briefly review these techniques and apply them to our design.

- **Interleaving Technique**
  
  Interleaving can be achieved by replacing the unit time, $z^{-1}$ in a system with $z^{-k}$, where $k$ is a positive integer. The interleaving technique is illustrated in Figure 3.10. Figure 3.10 illustrates a filter system before and after interleaving. In the top system $H(z)$ is the filter transfer function, $X(z)$ is the input signal and $Y(z)$ is the output signal without any interleaving. Figure 3.10 shows a $k$ interleaved filter system, where $H(z^k)$ is the filter transfer function, $X(z^k)$ is the input signal and $Y(z^k)$ is the output signal. The effect of interleaving is that it introduces $(k - 1)$ zeros between the input and the output signal. These
extra delays can be distributed within the system to achieve pipelining.

- **Delay Technique**
  The delay technique delays the output of the system by multiples of the unit time delay. Usually this delay is equal to the order of the filter for a FIR filter. These delays can be distributed uniformly within the system and can reduce the critical path of the design. For a DF FIR filter, the delays can linearly speed up the design at the cost of increased hardware latency.

Pipelining techniques can help reduce the critical path, eliminate broadcasting and global interconnections. However, these techniques need to be applied carefully to ensure the timing of the system is retained. One systematic way of applying these optimizations and retain the design timing is called cut-set retiming [32]. Cut-set retiming gives a set of guidelines to be followed while applying pipelining techniques within the design.

1. If a unit delay is added to the left arc we cut through the design and add a unit delay to all the left arcs and remove a delay from all the right arcs.

2. If there is no corresponding delay in the right arc to remove, we zero interleave the left arc by some factor 'k' and their corresponding inputs and outputs. After
interleaving, we add delay on the right arc and remove one corresponding delay in the left arc.

A similar procedure is followed if a delay is added to the right arc. For DSP designs we prefer the data flow in one direction since the data flow in multiple directions could lead to tighter constraints on the timing and will slow down the clock. Figure 3.16 and Figure 3.17 illustrate how cut-set retiming is applied on DF and TF-II FIR filters.

Data-Interleaving

The two optimization techniques suggested above, namely polyphase and pipelining can be applied to the DWT algorithm within each DWT block. The pipelining technique can also be applied to the system between each stage to speed up the design and reduce the critical path delay of the entire system. In this section we elaborate on the data-interleaving technique used in our design.

Data-interleaving helps process multiple independent signals using a single filter structure. If two signals are filtered independently by identical filters, we can use only one filter instead of two, interleave the inputs and add delays to the system. Figure 3.11(a) illustrates two different input sequences, \( x_1(n) \) and \( x_2(n) \) are filtered by identical filters \( H(z) \). The output sequences are \( y_1(n) \) and \( y_2(n) \) respectively.

We can zero-interleave the filter by a factor \( k = 2 \) for two signals by using the method described in pipelining section above. We could zero-interleave the filter by a factor \( k = n \) for \( n \) independent signals. The two input signals are interleaved and filtered as shown in Figure 3.11(b). Though interleaving by a factor \( k = 2 \) will double the registers in the filter, it will use the same number of multipliers and adders as required by a single filter. Hence, the resources required for multipliers and adders are shared. The output obtained \( y(n) \) is demultiplexed to obtain \( y_1(n) \) and \( y_2(n) \). The deinterleaved outputs have the same values as the outputs obtained by filtering
(a) Two independent signals filtered by two identical filters

(b) Interleaved signals filtered by single filter interleaved by $k = 2$

Figure 3.11: Data-interleaving
the input signals independently.

These kinds of optimizations can help reduce the resource utilization for a design that has similar filters for input signals. This results in a considerable savings in resources and efficient hardware utilization. However, interleaving a cascaded non-polyphase filter will reduce the throughput of the design, but if interleaving is combined with polyphase filters, it will maintain the throughput and also reduce the resource consumption. This is explained in Section 3.4.4. Therefore, data-interleaving technique encourages resource sharing and efficiently utilizes the hardware.

3.4 Hardware Design Implementation

3.4.1 Basic Implementation

The basic implementation of a 3-level 8-channel convolution based 1-D DWT algorithm is described in Chapter 2. The filter blocks are cascaded as shown in Figure 2.4. Further details of the inputs and outputs of each level and their frequency range is given in Section 2.2.1. Figure 3.12 illustrates the behavior of the input and output sequences for each level of decomposition. The length of the input and output samples have to be retained irrespective of the optimizations to maintain correct functionality of the DWT algorithm. In Figure 3.12, we show only one of the outputs of every level for illustration purposes. All the other outputs for a particular level of implementation are equivalent in length to the one illustrated.

The filtered output obtained at each level is downsampled by a factor of 2 as shown in Figure 2.4. The number of decimated output samples is half the number of input samples for each level since every even sample is retained and every odd sample discarded. The waveforms for level 1 downsampled output are shown in Figure 3.13. The control signal downsampler for level 1, is held high for alternate clock cycles. The downsampling logic is designed such that the time during which the downsampler is
Figure 3.12: IO waveforms for a DWT algorithm

high, the filtered output is sampled and the time when the downsampler is low the filtered output is discarded. Similar downsampling logic is implemented for higher levels of decomposition, where for level $J$ implementation the downsampler toggles every $2^J$th clock edge.

Figure 3.13: Decimation by two

The critical path delay for an $M^{th}$ order DF FIR filter for a 3-level implementation is $(T_{mult} + 3(M + 1)T_{add})$, which is undesirably large. The basic implementation in our design involves placing registers at the outputs of each level to restrict the critical path to $(T_{mult} + (M + 1)T_{add})$. The addition of these registers also saves the
intermediate outputs from each level. The LSBs of the final outputs of each level are truncated to maintain the same input length for all levels of implementations.

In the basic implementation, the outputs are downsampled after filtering. Consequently, every alternate sample is discarded which implies that half the computations were unnecessarily performed. The following implementation takes into account such wastage of computations and avoids this by using the polyphase technique.

3.4.2 Polyphase Implementation

The polyphase technique, as explained in Section 3.3.3, is applied to every decimated FIR filter within the DWT block to avoid the unnecessary calculations and to achieve higher computational efficiency. Figure 3.14 illustrates the 16-tap db8 low pass decomposition DF filter split into its equivalent polyphase structures. Filters $H_e$ and $H_o$ are even and odd polyphase filter structures. The input $x(n)$ is divided into its even and odd samples and filtered by the respective polyphase structures. The outputs of $H_e$ and $H_o$ are added to give the final output.

![Figure 3.14: Polyphase implementation for a level 1 low pass filter](image)

Figure 3.14 illustrates the equivalent waveforms for the level 1 polyphase filters.
The length of the input sample, output sample, intermediate signals and the number of delay units within the filter doubles as we go one level higher. However, the waveform pattern is similar in nature for higher levels of implementations. The relation between the signals in Figure 3.15 is:

\[ x(n) = \text{input signal} \]
\[ q(n) = x(n - 1); \text{delayed input signal} \]
\[ x_e(n) = \text{even samples of } x; \]
\[ x_o(n) = \text{odd samples of } x; \]
\[ Y_e(z) = X_e(z) H_e(z); x_e(n) \text{ filtered by } H_e \]
\[ Y_o(z) = X_o(z) H_o(z); x_o(n) \text{ filtered by } H_o \]
\[ y(n) = y_e(n) + y_o(n) \]

Figure 3.15: Polyphase implementation for a level 1 low pass filter

A lower number of computations results in power savings for the hardware architecture due to the use of polyphase structures. Furthermore, the critical path delay for a polyphase structure is reduced due to the reduction of the number of adders in the critical path when compared to a 16-tap DF FIR filter. In the next section
we explain how we pipeline the design to further reduce the critical path. However, the area required for a polyphase filter is slightly larger than the area for cascaded FIR filter due to an increase in the number of registers within the filter and one extra logic block to implement the downsampling process. The impact of using a polyphase structure in the DWT hardware design is that it reduces the power dissipated and achieves higher computational efficiency. These advantages outweigh the disadvantage of a slight increase in the hardware area.

### 3.4.3 Polyphase With Pipelined Implementation

Pipelining the filter structures helps speed up the design. Furthermore, applying pipelining techniques on the design reduces the critical path, eliminates broadcasting and global interconnections. In Section 3.3.3 we described various pipelining techniques that can be applied to various SFGs. We apply these pipelining techniques to the FIR filters within our design.

We illustrate how cut-set is applied on computational cells of the DF and the TF-II filter to pipeline the filters. Figure 3.16 illustrates the method by which we pipeline the computational cell of a DF filter. Variables $A$ and $B$ are the arcs that represent the data flow. The data flow for the computational cell of DF filter is in the same direction and we cut through arcs $A$ and $B$, as shown in Figure 3.16(a). We add single delays in both the arcs to obtain the final pipelined computational cell as shown in Figure 3.16(c).

Pipelining the DF filter reduces the critical path from $T_{\text{mult}} + (M - 1)T_{\text{add}}$ to $T_{\text{mult}} + T_{\text{add}}$ hence increasing the clock rate of the design. However, pipelining the design increases the hardware latency of the filter to $M$ clock cycles and also increases the number of registers required in the design.

In Figure 3.17, we illustrate how cut-set retiming is applied on the computational cell of a TF-II filter. This is an interesting example since the data flow of arcs $A$
(a) Intersect the SFG of the computational cell of a DF filter

(b) Add delays on both the arcs

(c) Final pipelined computational cell

Figure 3.16: Pipelined computational cell of a Direct Form (DF) Filter
and $B$ are in the opposite direction. For the computational cell presented in Figure 3.17(a), we cut through the arcs and interleave the registers in arc $A$ by a factor $k = 2$ to add a delay in arc $B$. This is done because adding a delay in arc $B$ without interleaving will result in elimination of the delay unit in arc $A$. The effect of this is data broadcasting, which is undesirable. After interleaving we obtain the TF-II computational cell as shown in Figure 3.17(b). However, decimated polyphase TF-II filters have computation cells identical to Figure 3.17(b) which makes the use of interleaving technique redundant. We can then safely add one delay in arc $B$ and remove a delay from arc $A$ as shown in Figure 3.17(b) to obtain the final pipelined architecture illustrated in Figure 3.17(c).

Pipelining the TF-II also reduces the critical path of the filter by the same factor as that of DF filters and it does not increase the latency of the design. The increase in area due to pipelining is smaller for TF-II compared to DF filters. However, data flow in opposite directions places tighter constraints on the clock and hence, results in slower clock speed compared to the DF filter.

We pipeline the polyphase filters presented in Section 3.4.2 for our design. It should be noted that the latency of the polyphase pipelined filter is half the latency of a cascaded filter (without polyphase) when pipelined in a similar manner. The final architecture for an 8-tap low pass pipelined DF and TF-II filters ($H_e$) are shown in Figure 3.18. These pipelining techniques are applied to all the filters in our DWT design.

We apply pipelining to the polyphase structure as a part of an incremental design process to help us understand how pipelining impacts the clock speed of the design. We illustrated how the pipelining of a DF filter increases its hardware latency whereas it does not change the latency of a TF-II filter. Also, the number of registers required for a TF-II structure is less as compared to DF filters. However, the clock frequency of the system would be higher for DF filters compared to TF-II filters. Hence for applications where latency is a critical design constraint, TF-II filters would be more
(a) Intersect the SFG of the computational cell of a TF-II filter

(b) Interleave the cell by a factor \( k = 2 \)

(c) Final pipelined computational cell

Figure 3.17: Pipelined computational cell of a Transpose Form-II (TF-II) Filter
(a) Direct Form

(b) Transpose Form-II

Figure 3.18: Pipelined structures for level 1 polyphase structure ($H_e$)
suitable. However, for applications that require higher throughput at the cost of extra registers and latency, DF filters would be more appropriate.

3.4.4 Hardware Implementation With Combined Optimizations

The architectures implemented in Section 3.4.2 and Section 3.4.3 present optimizations applied to the filter structures, namely polyphase and pipelining techniques. These techniques address the power and critical path delay issues for the DWT system. In this section, we present how we can further optimize the design at the architectural level.

The 3-level implementation of the DWT architecture is presented in Figure 3.19(a). \( H(z) \) and \( G(z) \) are the polyphase and pipelined lowpass and highpass filter structures respectively. The implementation in Figure 3.19(a) uses fourteen filter structures to implement a 3-level 8 channel DWT. For level \( J \), the number of filters required are \( 2^J \). However, for levels two and higher, we observe that multiple inputs are filtered through the same low pass and high pass filters. For example, \( y_L \) and \( y_H \) are filtered by the same filter \( G(z) \) in level 2. Hence, there are redundant filters within the design, two redundant filters for level 2 and six for level 3; that is \( (2^J - 2) \) redundant filters to each \( J \) level. We remove these redundant filters within the design by using data-interleaving techniques as described in Section 3.3.3. This results in significant savings in the hardware utilization. The architecture after implementing data-interleaving is illustrated in Figure 3.19(b).

Since \( y_L \) and \( y_H \) outputs of level 1 pass through the same filters, \( H(z) \) and \( G(z) \), we interleave these signals using a multiplexer and then proceed to filter them. This is similar to the method described in Section 3.3.3. In Figure 3.19(a), the outputs of the low pass filters in level 2 are \( y_{HL} \) and \( y_{LL} \). The same outputs are obtained in Figure 3.19(b) by splitting the interleaved output using a demultiplexer. Figure
(a) Basic 3-level 8 channel DWT structure with redundant filters

(b) Data-Interleaved structure

Figure 3.19: 3-level 8 Channel DWT structure with data-interleaving
3.19 gives an idea how data interleaving transforms the binary tree structure comprising of 14 filters into a structure comprising of 6 filters by using simple control logic.

The method described in Section 3.3.3 interleaved two signals using the basic cascaded filter structure which resulted in reducing the throughput of the system. When applying data-interleaving to the polyphase filters, there is no reduction in the throughput of the design caused by data-interleaving due to the nature of the polyphase filters. Figure 3.20 illustrates how interleaving is implemented for the 3-level 8 channel structure using polyphase filters.

![Figure 3.20: Our Method of implementation](image)

Details of the control blocks C1, C2, C3 and C4 are shown in Figure 3.21. The details in these blocks show the functionality implemented within the control structures. In Figure 3.20, C1, C2, C3 and C4 are the control blocks that interleave the input signal and downsamples it. In control block C1, the input $x$ is split into its even and odd indexed samples, $x_e$ and $x_o$ respectively as shown in Figure 3.21(a). The signals are then filtered using the even and odd polyphase structures to obtain the outputs of level 1, $y_L$ and $y_H$. These outputs are interleaved and downsamples in the control structure C2 as shown in Figure 3.21(b). The downsamples even and odd signals are $x_{2e}$ and $x_{2o}$ which are filtered once more using polyphase, pipelined and zero-interleaved filters in level 2. The interleaved outputs $y_{21}$ and $y_{22}$ of level 2 can be split into its individual outputs of level 2 as shown in Figure 3.21(c). These
Figure 3.21: Pipelined computational cell of a Transpose Form-II (TF-II) Filter
outputs are multiplexed and downsampled to obtain \( x_3_e \) and \( x_3_o \) which are the even and odd indexed inputs to level 3 polyphase structures. Figure 3.21(d) illustrates how the control block, C4, splits the outputs of level 3, namely \( y_{31} \) and \( y_{32} \) into the final outputs using demultiplexers. Figure 3.22 illustrates the SFG of the control logic used within C2. The select signal controls the multiplexers such that it correctly downsamples the input signals \( y_L \) and \( y_H \). The output \( y_H \) is delayed two clock cycles to achieve correct synchronization. The waveforms shown in Figure 3.23 illustrate the waveforms of the signals in Figure 3.22(b), which help us understand the control structure C2 better.

The outputs \( x_2_e \) and \( x_2_o \) are filtered by level 2 polyphase filters to obtain the outputs \( y_{21} \) and \( y_{22} \). Figure 3.24 illustrates how the output \( y_{21} \) is split to level 2 outputs \( y_{LL} \) and \( y_{HL} \).

Figure 3.25 illustrates the waveforms associated with Figure 3.24(b) which explains how the control signal fed into the demultiplexer assists in splitting the output \( y_{21} \).

Figure 3.22 and Figure 3.24 are simple yet efficient control structures that illustrate how interleaving with downsampling and demultiplexing can be implemented for any
Figure 3.23: Waveforms of control structure C2

Figure 3.24: Details of control demultiplexer
levels of decomposition. It should be noted that due to the use of polyphase filters on data-interleaving, the design does not slow down as it did when we used cascaded filters as shown in Figure 3.11. The data-interleaving method described above can be extended to higher levels of decomposition by adding control logic as presented in this section. Hence, the data-interleaving implemented in this architecture makes effective hardware utilization using simple control logic which is easily scalable to higher levels of implementation. Therefore, the final architecture shown in Figure 3.19(b) combines the polyphase, pipelining and data-interleaving optimizations into a single architecture. This architecture achieves low area, low critical path delay and less power utilization due to the use of the optimizations presented.

3.5 Chapter Summary

In this chapter, we discussed the design flow used to implement our design. We start by developing the basic software implementation of the algorithm in MATLAB and then compared the functionality of our hardware implementation with the MATLAB implementation to verify the design. We covered concepts of the hardware optimization techniques commonly used in DSP algorithms such as polyphase, pipelining and interleaving. The latter part of the chapter discussed the design techniques used to implement these optimizations on the DWT hardware.
We proposed an area efficient, high speed architecture that was developed from the basic binary tree architecture proposed by Mallat [10]. We also presented the combination of various optimization techniques on the hardware DSP algorithm. We presented a methodology that developed a basis for the application of these optimizations to higher levels of implementations. The optimizations were implemented at the filter level and the system design level. Filter level optimizations, namely polyphase and pipelining was applicable to many FIR filter structures which are modular in nature. We have implemented the DF and TF-II filters to illustrate how the choice of filter architecture can affect the performance of the design. The system design level optimizations such as data-interleaving were applicable to the DWT architecture due to its binary tree structure. A technique is presented that interleaves and downsamples within the same control structure. Therefore, the techniques presented in this Chapter makes it possible to easily scale the DWT algorithm for higher levels of implementation with minimum design complexity.
Chapter 4

Results and Analysis

In this chapter we present the results obtained by synthesizing the implementations presented in Chapter 3. We also analyze the impact of the optimizations on the performance of the DWT design on the Virtex-5 FPGA. We compare the architectures with respect to FPGA resource utilization, design throughput, power dissipated and hardware latency.

The chapter is divided into four sections. The first section discusses how we selected the computational precision required to implement the design in hardware. The next section describes the experimental setup and tools used to calculate the area, throughput and power for the designs. We then present the results obtained from the synthesis and analyze the results to select an implementation that performs well in terms of area, power, throughput and latency. The final section compares our hardware implementations to assess their performance results. We then select an architecture that performs well and we analytically compare it with several DWT implementations presented in the literature.
4.1 Bit Precision

Analysis of the effects of fixed point precision on the integrity of the output signals is a prerequisite to the Verilog hardware implementation of the DWT design. We compared the fixed point outputs using different word sizes for the inputs and filter coefficients to the floating point model of the DWT algorithm. The error metrics used for comparison are Mean Square Error (MSE) and Peak Absolute Error (PAE) which are defined in Equation 4.1.

\[
PAE = |y_{max} - \hat{y}_{max}|
\]

\[
MSE = \frac{1}{K} \sum_{i=1}^{K} (y[n] - \hat{y}[n])^2
\]

(4.1)

where \(y[n]\) and \(\hat{y}[n]\) are the floating point and fixed point numbers respectively. The maximum values for \(y[n]\) and \(\hat{y}[n]\) are \(y_{max}\) and \(\hat{y}_{max}\) respectively and \(K\) is the sequence length. The MSE is the mean of the cumulative squared error between the floating and fixed point numbers and PAE is the difference between the absolute maximum floating and fixed point numbers. These metrics help us assess the impact of scaling on the floating point numbers.

We assume that the inputs and the 16-tap Daubechies filter coefficients are scaled to \(N\) bits using the procedure described in Chapter 3. We use quantized speech signals as the input signal to test the sound quality and measure the effects of scaling. Table 4.1 summarizes the quantization errors for \(N = 8, 12\) and 16-bit input and coefficient word sizes.

From Table 4.1, we observe that the error metrics improved for higher bit precision but at the cost of increased resource utilization and power consumption. We reconstructed a sample speech signal wavefile in MATLAB to test the audio quality and to further assist us in selecting the bit precision. The audio quality for the speech signal reconstructed using 16 bits proved to be the best of the three and also appeared to have less noise. The audio from the 12-bit signal also had reasonably
Table 4.1: Quantization Results for Fixed-Point DWT implementation

<table>
<thead>
<tr>
<th>Input/Coeff Bits</th>
<th>Output Bits</th>
<th>PAE</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>16</td>
<td>6.15 * 10^{-1}</td>
<td>3.55 * 10^{-2}</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>2.78 * 10^{-2}</td>
<td>4.08 * 10^{-4}</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>1.45 * 10^{-3}</td>
<td>3.74 * 10^{-7}</td>
</tr>
</tbody>
</table>

good sound quality. The audio signal using 8-bits introduced quantization errors that made the quality of the reconstructed signal poor. Hence, we concluded that the 12-bit input was adequate for implementing the DWT algorithm using fixed-point computations and proceeded towards the Verilog hardware implementation and analyzed the performance of different DWT hardware architectures. However, computational bit precision could be easily varied depending upon the application requirements.

4.2 Virtex-5 FPGA

The Xilinx Virtex-5 FPGA is fabricated using the 65nm technology and is currently the latest FPGA family in the market. The Virtex-5 family is divided into four categories, each specialized for different applications. One of these categories is the Virtex-5 SXT family for high-performance signal processing applications. This FPGA family has a large number of logic cells, CLBs, embedded RAM and other state-of-the-art features. One of the most important features for our use is the DSP48E slices within the FPGA [5].

Most DSP applications require multiplier and accumulator blocks which when implemented on a traditional FPGA are slower, consume high logic utilization and dissipate more power. The Xilinx Virtex-5 FPGA family utilizes the DSP48E slices which are dedicated, low power slices, fully custom designed for DSP applications.
Each DSP48E slice has a dedicated multiplier and accumulator capable of operating at frequencies up to 550MHz. The DSP48E slices have other modes of operation where they can work as barrel shifters, adder chains and many more such modes. However, for our application we require them to function as multipliers. These slices have high speed, consume less area and power consumption compared to FPGAs without dedicated multipliers. Our basic implementation requires 224 multipliers and the Virtex-5 FPGA with the closest number of DSP48E slices we required was Xilinx Virtex-5 XC5VSX50T FPGA and therefore chosen for this work. During the course of our design and synthesis process we used commercially available tools such as the Xilinx ISE and the Xilinx XPower Estimator for Virtex-5 FPGA to estimate area, clock and power. The following two sections discuss how these tools are implemented.

4.2.1 ISE Project Navigator design flow

The Xilinx Integrated Software Environment (ISE) consists of a set of programs to capture, simulate and implement digital hardware designs in a FPGA [5]. We use the Xilinx ISE 9.2i Project Navigator GUI for this work to help us visually manage and automate the entire design process with the aid of toolbars, menus or icons. The design flow for implementing our design using the ISE project navigator is shown in Figure 4.1.

1. Design Entry
   We enter our Verilog design code using the ISE Text editor after we create a project and choose our FPGA device and package.

2. Behavioral Simulation
   We then perform a behavioral simulation of our Verilog hardware code to confirm the functional correctness of our design. The waveforms can be viewed in the graphical waveform viewer. The design can be simulated at any time during the design process to verify design functionality.
3. **Design Synthesis**

The synthesis process checks for syntax errors in the code and analyzes the hierarchy of the design. It then creates a netlist (.ngc) of primitive gates that contains the logical data and constraints. This could be implemented for any Xilinx FPGA device. We can specify the timing constraints; namely clock period, clock-to-pad and pad-to-clock, in the User Constraints file (.ucf file) for our design.

4. **Design Implementation**

The design is implemented using the following steps:

- **Translate**: The translate process runs NGDBuild that merges all the input netlists (the logical design data) and the design constraints information to create the ngd design file.

- **Map**: The map process maps the logic defined in the ngd file on the target FPGA. The output ncd file contains the information of the resources such
as the CLBs, registers and IOBs consumed on the target FPGA.

- **Place and Route**: Once the design is mapped, the design is then placed and routed (PAR) on the FPGA. This means that the resources described in the .ncd file are assigned specific locations on the FPGA. The connections between the resources are then mapped onto the FPGA’s programmable interconnect network. The PAR process has a large impact on the speed of the design and therefore it generates an updated ncd file with the static timing report. This report gives us the information for the clock frequency, the slack, whether there are any timing violations and the clock-to-pad and pad-to-clock delays.

The process of hardware implementation of the design on FPGA is completed if there are no errors and the timing constraints are met. The design summary generated after the complete process contains information of the resource utilization for the target FPGA and a detailed synthesis report. The design summary also displays the clock for the system in the clock report and the timing details are listed in the static timing report.

### 4.2.2 XPower Estimator

The XPower estimator (XPE) is a Microsoft Excel based spreadsheet developed by Xilinx Inc. for power estimation for certain Xilinx FPGA devices [33]. The XPE supports Virtex-5 FPGAs and we can use it in our predesign or preimplementation phase to get an estimate for the power consumed by our designs. The spreadsheet has multiple tabs such as the summary tab (default), clock tab, logic tab and DSP tab. We use macros to enter the power information in the calculation sheets. The formulas used for power calculations in the program are based on the intended behavior of our digital hardware design.

We need to enter the part, package and speed grades when we open the spreadsheet. The power consumption consists of two components: the static power and the
dynamic power. The static power, known as the quiescent power in the XPE, is the power consumed when there is no switching activity in the design. This depends primarily on the FPGA device selected. The quiescent power for our FPGA is 0.9W. The dynamic power is the power consumed by the load capacitance during switching activity. The formula for dynamic power calculations is shown in Equation 4.2 [34].

\[ P = CV^2EF \]  

where, \( P \) is the power in mW, \( C \) is the load capacitance in Farads, \( E \) is the switching activity for the element and \( F \) is the frequency of operation in Hz. The XPE requires design information such as the resource utilization (LUTs, IOs, registers, DSPs), clock frequency, clock fanout and the toggle rates to estimate the dynamic power for our design. We can either manually populate the spreadsheet or import the map report file (.mrp) from the ISE project navigator to get the actual resource utilization for a given design. We need to have a good understanding of our design to get an estimate for the toggle in our design. We can obtain the rest of the information such as the clock frequency and clock fanout of our design from the design summary and the reports generated by Xilinx ISE.

4.3 Results and Analysis

In this section, we illustrate the performance results of the architectures we synthesized on the Virtex-5 FPGA and we discuss the area, clock, latency and power results.

4.3.1 Hardware Design Architectures

The design space considered in this work consists of ten hardware implementations listed in Table 4.2. The basic architectures DF_basic and TF-II_basic implement a design similar to Baganne [13] with an added optimization of pipelining
between the stages of decomposition (an optimization similar to Marino [1]). Architectures, DF\text{.poly} and TF-II\text{.poly}, implement an optimization that considers reducing the power dissipation ideally accomplished using the polyphase decimated filter implementation. The effect of data-interleaving is considered as a design technique to reduce the hardware resource utilization of the FPGA. The polyphase architectures that implement data-interleaving are DF\text{.poly+intr} and TF-II\text{.poly+intr}. We pipeline the DF and TF-II filter structures in the polyphase architectures and the polyphase with data-interleaved architectures to assess how pipelining impacts the throughput of the design. The polyphase with pipelining architectures are DF\text{.poly+pipe} and the TF-II\text{.poly+pipe} for DF and TF-II filters respectively. The architectures that consider all the combined optimizations are DF\text{.poly+pipe+intr} and the TF-II\text{.poly+pipe+intr}.
designs. We have considered the DF and TF-II filter structures for all the design optimizations to illustrate how filter structures can affect the latency, area, power and the throughput of the design. All ten designs have been synthesized on the Virtex-5 FPGA. This helps us get a good assessment of the various factors that affect the performance for the DWT design.

4.4 Hardware Synthesis Results

The basic metrics we considered in the implementation of the convolution-based DWT are area, critical path delay, latency and power. In this section, we present the results we obtained after synthesizing our designs on the Virtex-5 FPGA.

4.4.1 Area

The area analysis is the resource utilization for the FPGA. The main factors that we have considered which contribute to the resource utilization are the number of registers, LUTs and DSP48E slices for the Xilinx Virtex-5 FPGA. These results are obtained from the Xilinx ISE design summary for each of our designs. Table 4.3 displays the number of registers, LUTs and DSPs consumed by each of our designs. The % column next to each result displays the total percentage of the resources consumed by the Virtex-5 FPGA.

Figure 4.2 illustrates a graph of the normalized area. The resources are normalized individually for registers, LUTs and DSP48Es. We observe that the polyphase structures have twice the number of registers compared to the basic implementation. This is due to the doubling of the delay units within the polyphase subfilters. However, the number of LUTs and the DSP48Es for the polyphase structures are the same as that for the basic implementation. The DF$_{poly+pipe}$ implementation has approximately four times the registers of the basic implementation whereas the TF-II$_{poly+pipe}$ has approximately 2.5 times more registers. We observe that the least hardware con-
<table>
<thead>
<tr>
<th>Design Name</th>
<th>Registers</th>
<th>% of</th>
<th>LUTs</th>
<th>% of</th>
<th>DSP48Es</th>
<th>% of</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF_basic</td>
<td>3174</td>
<td>9</td>
<td>26233</td>
<td>80</td>
<td>224</td>
<td>77</td>
</tr>
<tr>
<td>DF_poly</td>
<td>5898</td>
<td>18</td>
<td>26564</td>
<td>81</td>
<td>224</td>
<td>77</td>
</tr>
<tr>
<td>DF_poly+pipe</td>
<td>13619</td>
<td>41</td>
<td>21608</td>
<td>66</td>
<td>224</td>
<td>77</td>
</tr>
<tr>
<td><strong>DF_poly+intr</strong></td>
<td><strong>5610</strong></td>
<td><strong>17</strong></td>
<td><strong>10383</strong></td>
<td><strong>31</strong></td>
<td><strong>96</strong></td>
<td><strong>33</strong></td>
</tr>
<tr>
<td>DF_poly+pipe+intr</td>
<td>8754</td>
<td>27</td>
<td>9307</td>
<td>28</td>
<td>96</td>
<td>33</td>
</tr>
<tr>
<td>TF-II_basic</td>
<td>3148</td>
<td>9</td>
<td>23489</td>
<td>71</td>
<td>224</td>
<td>77</td>
</tr>
<tr>
<td>TF-II_poly</td>
<td>5870</td>
<td>17</td>
<td>23429</td>
<td>71</td>
<td>224</td>
<td>77</td>
</tr>
<tr>
<td>TF-II_poly+pipe</td>
<td>8222</td>
<td>25</td>
<td>21581</td>
<td>66</td>
<td>224</td>
<td>77</td>
</tr>
<tr>
<td><strong>TF-II_poly+intr</strong></td>
<td><strong>5586</strong></td>
<td><strong>17</strong></td>
<td><strong>10087</strong></td>
<td><strong>30</strong></td>
<td><strong>96</strong></td>
<td><strong>33</strong></td>
</tr>
<tr>
<td>TF-II_poly+pipe+intr</td>
<td>8274</td>
<td>25</td>
<td>9295</td>
<td>28</td>
<td>96</td>
<td>33</td>
</tr>
</tbody>
</table>
Figure 4.2: Normalized Area Graph
sumption is by the poly+intr filters. They have 57% less DSP48E slices compared to other architectures. This is a significant reduction since there are limited number of DSP48 slices on an FPGA and should be prudently used. The number of registers for these structures is similar to polyphase structures in spite of reduced filters since the registers within the filters are increased due to data-interleaving. Pipelining the polyphase with interleaved structures again increases the registers in the design. However, an increase in number of registers may be necessary for applications that require higher throughput. From our table and graph, we can conclude that the DF\textunderscore poly+intr and TF-II\textunderscore poly+intr have the least overall resource consumption. The second most area efficient architecture is TF-II\textunderscore poly+pipe+intr followed by the DF\textunderscore poly+pipe+intr architecture. The remaining architectures are not very area efficient. Therefore, data-interleaving is the main influencing factor in reducing the resource consumption for the FPGAs.

4.4.2 Throughput

The pipelining technique reduces the critical path delay and increases the clock frequency and the throughput of the design. Throughput is the number of samples obtained per second. Number of samples obtained depends on the toggle rate of the system. Therefore, throughput can be expressed as illustrated in Equation 4.3.

\[ Throughput = \frac{\text{Toggle rate}}{\text{Delay (sec)}} \]  \quad (4.3)

The toggle rate for level 3 outputs is 12.5%, that is 0.125 for all the designs. We observe in Table 4.4 that for the pipelined designs, reduction in the delay causes a corresponding increase in throughput. The DF\textunderscore poly+pipe+intr and TF-II\textunderscore poly+pipe+intr architectures have registers in the control logic which adds to additional reduced critical path and further increases the throughput. However, we observe that the throughput of the DF pipelined structures is higher than the TF-II filters. This is because the data flow of the inputs and the outputs is in opposite directions. This introduces added constraints to the clock of the system and hence slows down the
system. The maximum throughput is exhibited by \texttt{DF Poly+Pipe+intr}.

Table 4.4: Throughput Results

<table>
<thead>
<tr>
<th>Design Name</th>
<th>Clock (MHz)</th>
<th>Delay (ns)</th>
<th>Throughput (MSamples/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF basic</td>
<td>50.0</td>
<td>20.0</td>
<td>6.250</td>
</tr>
<tr>
<td>DF poly</td>
<td>50.0</td>
<td>20.0</td>
<td>6.250</td>
</tr>
<tr>
<td>DF poly+pipe</td>
<td>73.5</td>
<td>13.6</td>
<td>9.191</td>
</tr>
<tr>
<td>DF poly+intr</td>
<td>50.0</td>
<td>20.0</td>
<td>6.250</td>
</tr>
<tr>
<td>\texttt{DF Poly+Pipe+intr}</td>
<td><strong>82.9</strong></td>
<td><strong>12.1</strong></td>
<td><strong>10.417</strong></td>
</tr>
<tr>
<td>TF-II basic</td>
<td>50.0</td>
<td>20.0</td>
<td>6.250</td>
</tr>
<tr>
<td>TF-II poly</td>
<td>50.0</td>
<td>20.0</td>
<td>6.250</td>
</tr>
<tr>
<td>TF-II poly+pipe</td>
<td>67.0</td>
<td>14.9</td>
<td>8.333</td>
</tr>
<tr>
<td>TF-II poly+intr</td>
<td>50.0</td>
<td>20.0</td>
<td>6.250</td>
</tr>
<tr>
<td>\texttt{TF-II Poly+Pipe+intr}</td>
<td><strong>73.6</strong></td>
<td><strong>13.5</strong></td>
<td><strong>9.259</strong></td>
</tr>
</tbody>
</table>

4.4.3 Latency

Hardware latency for our design is determined by the number of clock cycles between the inputs and output samples. We measure the difference in clock cycles between the MATLAB generated waveforms (reference waveforms) and the waveforms generated by our Verilog designs to determine the latency for our designs. Table 4.5 tabulates the latency for our designs.

We observe from Table 4.5 that the latency of the pipelined DF filters is higher
Table 4.5: Latency Results

<table>
<thead>
<tr>
<th>Design Name</th>
<th>Latency (clock cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF_basic</td>
<td>4</td>
</tr>
<tr>
<td>DF_poly</td>
<td>4</td>
</tr>
<tr>
<td>DF_poly+pipe</td>
<td>21</td>
</tr>
<tr>
<td>DF_poly+intr</td>
<td>7</td>
</tr>
<tr>
<td>DF_poly+pipe+intr</td>
<td>24</td>
</tr>
<tr>
<td>TF-II_basic</td>
<td>4</td>
</tr>
<tr>
<td>TF-II_poly</td>
<td>4</td>
</tr>
<tr>
<td>TF-II_poly+pipe</td>
<td>4</td>
</tr>
<tr>
<td>TF-II_poly+intr</td>
<td>7</td>
</tr>
<tr>
<td>TF-II_poly+pipe+intr</td>
<td>7</td>
</tr>
</tbody>
</table>

than the TF-II filters. The increase in latency for interleaved structures is due to the added registers in the control logic. Hence, for applications where latency is a critical design constraint, pipelined DF filters would not be suitable.

4.4.4 Power

We estimate the power consumed for each of our designs using the XPower estimator as explained in Section 4.2.2. We need multiple voltage supplies to power our FPGA. We have used the default values in the XPE: $V_{ccint}$ 1V and $V_{ccaux}$ 2.5V. The IO pins consume a large power since these are designed in a larger geometry.
than the core to support sinking currents for all of the IO standards (courtesy: Tony Scarangella, Application Engineer, Xilinx Inc.). However, since all our designs have the same IO pins, the power consumed would be similar. IO power is approximately 0.06W for our designs.

The Logic tab in the XPE accounts for the power consumed by the CLBs and the registers within our design. The DSP tabs contain information about the number of DSP48Es used in our design. We use the waveform simulations generated by the Xilinx Project Navigator to estimate the toggle rates for our design. Toggle rates reflect how often an output changes relative to clock. If the output changes at every rising edge of the clock the toggle rate is 100%. Figure 4.3 illustrates the toggle rates for different signals. Table 4.6 and Figure 4.4 compare the power consumed by our designs. The total DSP slices consumed in our basic design are 224. Level 1 use 32 DSP slices, level 2 use 64 and level 3 use 128 slices; a ratio of 1:2:4. The signals in level 1 switch at every rising clock edge, hence the toggle rate for level 1 is 100%. After decimation, the input to level 2 is two clock cycles long, hence all the signals toggle at 50%. Similarly, the toggle rate for level 3 is 25%. Toggle rates of 100%, 50% and 25% is illustrated in Figure 4.3. The power consumed by all the levels of the basic implementation for a 50MHz clock is equal to 0.058W, this adds up to a

Figure 4.3: Toggle rates
total power consumption of 0.174W by the DSP48Es.

The toggle rates for the polyphase decimated filters, DF\textsubscript{poly} and TF\textsubscript{poly}, is half as that of the basic architecture (50%:25%:12.5%) since the input signals are downsampled by two prior to filtering for the polyphase filters. The DSP slice distribution for polyphase filters is same as that of the basic design. However since the toggle rates for the polyphase design is half, the power consumed by the DSP48E slices is also half as seen in Table 4.6. However, these toggle rates do not reduce the power consumed by the registers since the registers in the polyphase filters are almost twice the registers in the basic architecture. We hence conclude that the total power consumption for polyphase decimated structures is less than architectures with non-polyphase filters. This is illustrated by the power graph in Figure 4.4.

The power consumed by the DSP48Es for polyphase with interleaved structures is similar to the power consumed by the polyphase filters in spite of reduced area. The reason being, the toggle rates for all the signals in all levels of implementation is the same due to interleaving. From Table 4.6, we also observe that the power consumed by the pipelined structures is higher than the non-pipelined structures. This is because pipelining increases the number of registers in the design and also increases frequency of the design. Power is proportional to frequency as shown in Equation 4.2, hence the power for pipelined filters increases at constant voltage and toggle rates. From Table 4.6 and Figure 4.4, we conclude that the designs that consume the least power are the polyphase with interleaved structures followed by the architectures with all the combined optimizations. TF-II\textsubscript{poly}+pipe followed by TF-II\textsubscript{poly}+pipe+intr design has the least power consumption from the set of pipelined designs.
Table 4.6: Power Results for DWT architectures in design space

<table>
<thead>
<tr>
<th>Design Name</th>
<th>Registers (W)</th>
<th>LUTs (W)</th>
<th>DSP48Es (W)</th>
<th>Toggle Rates level (1:2:3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF_basic</td>
<td>0.018</td>
<td>0.076</td>
<td>0.174</td>
<td>100:50:25</td>
</tr>
<tr>
<td>DF_poly</td>
<td>0.018</td>
<td>0.067</td>
<td>0.087</td>
<td>50:25:12.5</td>
</tr>
<tr>
<td>DF_poly+pipe</td>
<td>0.06</td>
<td>0.12</td>
<td>0.127</td>
<td>50:25:12.5</td>
</tr>
<tr>
<td>DF_poly+intr</td>
<td><strong>0.039</strong></td>
<td><strong>0.046</strong></td>
<td><strong>0.087</strong></td>
<td><strong>50:50:50</strong></td>
</tr>
<tr>
<td>DF_poly+pipe+intr</td>
<td>0.1</td>
<td>0.068</td>
<td>0.143</td>
<td>50:50:50</td>
</tr>
<tr>
<td>TF-II basic</td>
<td>0.018</td>
<td>0.078</td>
<td>0.174</td>
<td>100:50:25</td>
</tr>
<tr>
<td>TF-II_poly</td>
<td>0.018</td>
<td>0.067</td>
<td>0.087</td>
<td>50:25:12.5</td>
</tr>
<tr>
<td>TF-II_poly+pipe</td>
<td>0.033</td>
<td>0.063</td>
<td>0.116</td>
<td>50:25:12.5</td>
</tr>
<tr>
<td>TF-II_poly+intr</td>
<td><strong>0.039</strong></td>
<td><strong>0.046</strong></td>
<td><strong>0.087</strong></td>
<td><strong>50:50:50</strong></td>
</tr>
<tr>
<td>TF-II_poly+pipe+intr</td>
<td>0.081</td>
<td>0.06</td>
<td>0.126</td>
<td>50:50:50</td>
</tr>
</tbody>
</table>
Figure 4.4: Power Graphs
4.5 Architectural Analysis

We analyzed the results for the various performance parameters individually in the previous section. In this section we compare our architectures to determine which architectures perform efficiently for different design constraints. We also present an analytical comparison of our architecture to the some of the 1-D DWT architectures already implemented in the past.

4.5.1 Comparison of Architectures

Comparison between different architectures, opens up an avenue for future reuse of our hardware for application specific DWT architectures. We can select the architectures that are best suited for area, power, clock and latency using the results in Section 4.4. The DF_poly+intr and TF-II_poly+intr architectures are best suited for applications where the resource utilization and power dissipation are primary concerns. The throughput and the latency for these architectures also perform fairly well. The implementations with combined optimizations are additional DWT structures that perform well. The DF_poly+pipe+intr architecture has slightly larger area and power consumption than TF-II_poly+pipe+intr. The latter architecture also has significantly less latency compared to the DF architecture with the combined optimizations. However, the throughput for DF_poly+pipe+intr is higher than that of TF-II_poly+pipe+intr. Therefore, DF_poly+intr and TF-II_poly+intr would be good choices for applications such as hand held or portable devices where area and power dissipation are important performance criterions. The DF_poly+pipe+intr would be the most suitable architecture for stand-alone applications, where high throughput is the primary concern at an additional penalty of area, power and latency increase. Finally, TF-II_poly+pipe+intr would be a suitable choice for most applications, where at the cost of slight increase in area over the non-pipelined designs we can achieve good throughput at the cost of minimal additional latency. Therefore, TF-II_poly+pipe+intr architecture had the best combination of all the performance metrics.
Different stages of pipelining can be applied depending on the design requirements and the applications. We could choose pipelined multipliers with different depths to further increase the throughput of the design. Alternately, we could selectively pipeline the filter structures in the design to increase the throughput without sacrificing much of the area or the power. Therefore, a tradeoff between all the parameters can be easily achieved depending on the application.

4.5.2 Analytical Comparison of DWT Architectures

We perform an analytical comparison of the TF-II\textsubscript{poly+pipe+intr} architecture to the architectures we described in Section 2.2.2. This gives us a further assessment of how our architecture performed as compared to the ones proposed in the literature. Table 4.7 compares the architectures with respect to resource utilization, critical path and the number of clock cycles required per computation. The proposed architecture we used for comparison is the TF-II\textsubscript{poly+pipe+intr} since it performs well in terms of area, power, latency and throughput.

Our architecture performed considerable well in terms of resource utilization and critical path over Baganne’s architecture [13]. The folded architectures proposed by Denk [2] and Premkumar [18] consumed less resources but required more clock cycles to perform a single computation. Zhang’s architecture [3] was an improvement over the folded architectures that required a single processor [2] [18], but it required twice as many resources. Additionally, the feedback loops in Zhang’s architecture eventually slowed down the clock of the system. Although our architecture required more resources compared to Zhang, our architecture required half the computational time and was comparatively easily scalable to higher levels of decomposition. Marino’s architecture [1] employed coefficient interleaving and was closest to our design in terms of resource utilization and critical path. However, Marino’s design required twice the time to compute one value and had additional drawbacks of increased switching between the coefficients and discarding of outputs for levels higher than two, hence
Table 4.7: Performance comparison for 16-tap, 3-level DWT Designs

<table>
<thead>
<tr>
<th>DWT Design</th>
<th>No. of Mults.</th>
<th>No. of Adders</th>
<th>No. of Regs.</th>
<th>Computational cycles</th>
<th>Critical Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baganne [13]</td>
<td>224</td>
<td>210</td>
<td>105</td>
<td>1</td>
<td>$T_{\text{mult}} + 15T_{\text{add}}$</td>
</tr>
<tr>
<td>Denk [2]</td>
<td>32</td>
<td>32</td>
<td>96</td>
<td>4</td>
<td>$T_{\text{mult}} + T_{\text{add}}$</td>
</tr>
<tr>
<td>Marino [1]</td>
<td>96</td>
<td>90</td>
<td>224</td>
<td>2</td>
<td>$T_{\text{mult}} + T_{\text{add}}$</td>
</tr>
<tr>
<td>Premkumar [18]</td>
<td>32</td>
<td>30</td>
<td>472</td>
<td>4</td>
<td>$T_{\text{mult}} + T_{\text{add}}$</td>
</tr>
<tr>
<td>Zhang [3]</td>
<td>64</td>
<td>60</td>
<td>120</td>
<td>2</td>
<td>$T_{\text{mult}} + T_{\text{add}}$</td>
</tr>
<tr>
<td>Proposed</td>
<td>96</td>
<td>90</td>
<td>448</td>
<td>1</td>
<td>$T_{\text{mult}} + T_{\text{add}}$</td>
</tr>
</tbody>
</table>

reduced efficiency. Our DWT architecture exhibited reasonable resource utilization, required only one computational cycle per value, has small critical path and required fairly low design complexity.

4.6 Chapter Summary

In this chapter, we analyzed the performance of ten 1-D DWT design architectures and compared the performance results of the architectures with respect to resource utilization, power, latency and area. The effects of the polyphase, pipelined and interleaved optimizations of the DWT design were displayed and analyzed. We discovered design tradeoffs which make our design flexible depending on the application needs in our attempt to create an architecture that is efficient in terms of area, power, latency and throughput.

The results displayed were obtained by synthesizing these designs onto a Virtex-5
FPGA. Other FPGAs result in different performance results. However, the method we used to develop a design that performed well can be applied to different FPGAs. The process for measuring the basic performance metrics would remain the same for the different FPGAs as long as we used the same Xilinx ISE design kit.
Chapter 5

Conclusion and Future Work

5.1 Conclusion

This work presents a design methodology for the implementation of a multi-level 1-D discrete wavelet transform on a FPGA using a combination of essential hardware optimization techniques. The optimizations considered for this work are data-interleaving for reduced hardware utilization and polyphase filter structures coupled with pipelining to achieve low power and high-throughput respectively. The design techniques presented systematically combined these optimizations to develop a flexible DWT architecture that is efficient in terms of FPGA resource utilization, power dissipation, design throughput and hardware latency.

We implemented these optimization for a 3-level, 8-channel DWT hardware architecture using the DF and TF-II filters and mapped ten unique DWT designs onto the Xilinx Virtex-5 FPGA. Through an incremental design and analysis process we derived an efficient hardware DWT architecture and concluded that the TF-II FIR filter with combined optimizations exhibited the best combination of performance metrics, namely resource utilization, throughput, latency and power dissipation in
the design space. We further discussed the fundamental design issues and the results obtained from the synthesis of other architectures in the design space on the FPGA. We analyzed these results and presented the advantages, limitations and different performance parameter tradeoffs associated with these optimizations.

Most of the efficient DWT implementations presented in the past have targeted specific design specifications which exhibited good efficiency for only one or two of the performance parameters and were suitable only for certain filter structures. Whereas, the architectures presented in this work took into account multiple performance metrics, exhibited low design complexity and were fairly modular in nature. Furthermore, our architectures were flexible and easily scalable to varying filter lengths, filter structures, levels of decomposition and design specification needs. The optimizations implemented were intended to make the architecture suitable for most of the portable, high speed, power-efficient devices.

The critical performance criteria vary with the need of the application. The DWT is a major block suitable for many multimedia applications used in various platforms such as mobile phones, HDTV, printers, GPS systems, video systems, internet and many more. Therefore, the availability of an efficient, yet flexible architecture, that can easily adapt to the different system requirements was essential. The importance of our work lies in the method by which we systematically combined all the optimizations for an efficient implementation of the DWT algorithm and analyzed the different performance tradeoffs. The design approach used for this work could be helpful to DSP hardware developers who are required to implement the DWT to meet specific design constraints. With the aid of our work, the designers can assess which optimization would be best suited to their application requirements and what other design tradeoffs would they have to consider. This would save the hardware developers a lot of trial-and-error effort and would help them achieve their application specific DWT architecture faster. The optimizations and architectures presented could also be implemented on Application Specific Integrated Circuits (ASICs) for higher performance needs. However, we have implemented these architectures on re-
configurable hardware such as FPGAs to maintain the flexibility for the application. Additionally, the FPGAs have shorter development cycle and lower NRE costs. The new generation FPGAs, such as the one used in our work, are capable of performing extremely well due to the in-built DSP chips tailored to meet the requirements of high performance signal processing.

5.2 Future Work

The focus of this thesis was on the implementation and optimization of the convolution-based 1-D DWT decomposition hardware architecture. Considerable amount of work has been done in other hardware DWT architectures where these design techniques and optimizations presented by our work can be extended to help make incremental improvements in their hardware performance. Some of these are listed below.

1. 2-D DWT

The 2-D DWT is used by JPEG2000 for image compression and can be implemented on an image by succession of row-wise and column-wise 1-D filtering. This leads to 4-band per resolution level decomposition as demonstrated in Figure 5.1. Higher levels of decomposition can be achieved by iterating the 2-D filtering over each band. The 2-D DWT has higher computational complexity and requires more filters for each level of implementation compared to the 1-D DWT architecture. Therefore, for future work an effective methodology to optimize the 2-D DWT hardware architecture could be explored using the methodology we developed as a basis.
2. Lifting-based DWT

Another popular alternative to the convolution based DWT is the lifting-based scheme. We briefly described the lifting based scheme in Section 2.2.3. Future goals would include analyzing the impact of the convolution optimizations on the performance improvement of the lifting-based DWT architecture.

3. Reconstruction Filters

We presented the subband decomposition of the 1D DWT for this work. The subbands can be reconstructed using synthesis filters. This would require interpolation filter instead of decimation filters. The methodology developed for the 1D DWT decomposition can be extended to the reconstruction process. This is possible since the signal reconstruction is a reverse design technique as Figure 5.2 illustrates for the 3-level 8-channel synthesis architecture.
Figure 5.2: 3-level 8-channel Reconstruction Filter [6]
Bibliography


