MIPS Augmented with Wavelet Transform: Performance Analysis

By: Prarthana Shrestha

Computer Engineering Laboratory
Faculty of Information Technology and Systems
Delft University of Technology
The Netherlands

July 2002

Delft University of Technology
Faculty of Information Technology and Systems

Type:	Master's Thesis
Number of pages:	51
Date:	12 July, 2002

Lab./Dept.	Laboratory of Computer Engineering
Code number:	1-68340-28 (2002)-02
Author:	Prarthana Shrestha
Title:	MIPS Augmented Wavelet Transform: Performance Analysis
Supervisor:	Prof. S. Vassiliadis
Mentors:	G. Kuzmanov
	J.S.S.M Wong

Abstract

Wavelet transform provides a natural way for building a reversible and irreversible image compression system. Lifting scheme, a key feature of the JPEG-2000 standard, is the fastest implementation of wavelet transform. However, a pure software implementation of lifting wavelet transform is considered to be a substantial bottleneck for the system using it. In this thesis, we employed and compared 3 software approaches that implement the Fast Lifting Wavelet Transform (FLWT), namely; modified version of Liftpack, the reversible wavelet transform (Le Gall 3-5filter) and the irreversible wavelet transform (Daubechies 9-7). All these approaches are based on integer wavelet transforms using lifting scheme on two-dimensional images but differed in filtering methods and boundary treatment of the data. Out of this comparison, the modified version of the Liftpack was found to be the fastest FLWT implementation. We further investigate the software only implementation of FLWT with a hybrid software/hardware implementation. A reconfigurable hardware implementation of the FLWT algorithm was simulated as a functional unit in a MIPS based processor. A new instruction was introduced as an ISA extension to the MIPS architecture for the FLWT reconfigurable hardware unit. Simulations were carried out in a cycle accurate simulator 'Sim-outorder'of the SimpleScalar Toolset (V 3.0). For an image size of 352*288,the results indicate a speed up of over 4 times for Liftpack , 3.7 times for the reversible wavelet transform and 11 times for the irreversible wavelet transform versus the pure software implementation. It was also noted that the speed-up rises with increase in the picture dimensions or increase in the length of the filter.

Acknowledgement

Firstly, I would like to express my sincere gratitude to Professor S.Vassiliadis for allowing us to work under his supervision and for all his support and suggestions.

Special thanks to G. Kuzmanov and J.S.S.M. Wong, mentors of this thesis, for giving their valuable time and co-operation throughout the way.

I would also like to thank Pyrrhos Stathis for timely helps regarding SimpleScalar simulator.

Lastly, I would like to thank all of my friends who helped making this project a nice experience.

Contents

Abstract……………….……………………………………………………………………...i

Acknowledgement……………………………………………………………….……..ii

Contents…………………………………………………………….………….……iii

List of Figures..…………………………………………………….…………………..v

List of Tables…………………………………………………………………...……vi

1 Introduction..1

1.1 Objective and Framework 1
1.2 Organization of the Thesis 2

2 Image Compression...3

2.1 Digital Images and Compression...3
2.2 Compression Techniques 3
2.2.1 Transform Coding 4
2.2.2 Quantization...5
2.2.3 Variable Length Encoding 5

3 Wavelet Transform 6

3.1 Wavelets 6
3.2 Multi Resolution Analysis 7
3.3 Generating Wavelets Using Two-Channel Filterbanks 8
3.4 The Fast Wavelet Transform 9
3.5 Advantages of Using Wavelets 10

4 Lifting..11

4.1 Lifting Scheme 11
4.1.1 Split Phase...12
4.1.2 Predict Phase (Dual Lifting) 12
4.1.3 Update Phase (Primal Lifting) 14
4.2 Inverse Transform 14
4.3 Integer-to-Integer Transform...15
4.4 Advantages of the Lifting Scheme 16
4.5 Boundary Treatment..16
4.5.1 Classical Extension Methods 16
4.5.2 Boundary Filtering 18

5 Software Implementation of Selective Lifting Schemes....................................19

5.1 Liftpack 19
5.1.1 Filter Implementation...19
5.1.2 Boundary Treatment 21
5.1.3 Transform Algorithm 22
5.1.4 Modifications on Liftpack Source Code 23
5.2 Reversible Wavelet Transform...23
5.2.1 Boundary Treatment 23
5.2.2 Filter Procedure..24
5.2.3 Transform Algorithm 24

5.3 Irreversible Wavelet Transform 25
5.3.1 Boundary treatment 25
5.3.2 Filter Procedure 25
5.3.3 Transform Algorithm 26

6 Lifting Scheme in Reconfigurable Unit 28

6.1 Reconfigurable Computing 28
6.1.1 Introduction 28
6.1.2 Placement of Reconfigurable Unit 29
6.2 Wavelet Transform in Reconfigurable Unit 30

7 Methodology 31

7.1 SimpleScalar Architectural Design Tool Set 31
7.1.1 SimpleScalar 31
7.1.2 Sim-Outorder 32
7.2 MIPS Augmented with Reconfigurable Unit 33
7.3 Instructions Annotation for FLWT Operation 34
7.4 FLWT Implementation 35
7.5 Modifications on Sim-outorder Source Code 35
7.6 The Memory Simulation 36
7.7 Benchmarks 37

8 Simulation Results Analysis 38

8.1 Simulation Metrics 38
8.1.1 Calculation for Software Execution Time 38
8.1.2 Calculation for Hardware Execution Time 39
8.2 Simulation Results 39
8.2.1 Performance Analysis of Liftpack with Different Picture Sizes 39
8.2.2 Performance Analysis of Liftpack with Different Polynomial Filters 42
8.2.3 Comparative Analysis of Different Filter Implementations 43
8.3 Effect of Latency on the Performance of Hardware Unit 44
8.4 Transformation Accuracy 46

9 Conclusions and Suggestions for Future Research 47

Reference…....................49

Appendix A…....................50

List of Figures

Figure 2.1: Block representation of digital image compression system ………..…………….4
Figure 2.2: Transform coding ……………………………………………...…...…………….4

Figure 3.1: Wavelet prototype function ………………………………………..….……….6
Figure 3.2: Illustration of wavelets at different scales …….………………………………….7
Figure 3.3: Multiresolution analysis on wavelet transform ……………………..…...…….7
Figure 3.4: Iterated filter bank ……………….…………………………………………...…..8
Figure 3.5: The subband labeling scheme for one level, 2-D transform ……..…………..…9
Figure 3.6: Labeling the 2 level 2-D transform …………………..………..……..………9
Figure 3.7: Original Lena image ……………………………...………………………9
Figure 3.8: 2 level, 2-D wavelet transform ……………………………………………….9

Figure 4.1: The Lifting Scheme: Split, Predict and Update ……………………………....11
Figure 4.2: Linear interpolation, N=2 ……………………………………...…….….……13
Figure 4.3: Cubic interpolation, N=4 ……………….……………………...……………..13
Figure 4.4: The Lifting Scheme: Predict γ-1,k and update λ-1,k ……………………………13
Figure 4.5: Number of samples in different levels ……………………………………….14
Figure 4.6: The Lifting Scheme, inverse transform: Update, Predict and Merge ……..…….14
Figure 4.7: A finite signal………………………………………………...…………….17
Figure 4.8: Periodic extension of signal ……………………………………...………..17
Figure 4.9: Periodic symmetric extension (1,1) of the finite length signal…………………17
Figure 4.10: Cases of symmetrical extension using lifting …...…………………………....18

Figure 5.1: Behavior of the cubic interpolating subdivision near the boundary…………..…21
Figure 5.2: Periodic symmetric extension of signal……………………………………..…23

Figure 6.1 Different levels of coupling in a reconfigurable system………..……………….29

Figure 7.1 Architecture of SimpleScalar…………………………………………………31
Figure 7.2: SimpleScalar Architecture instruction formats………………………………….32
Figure 7.3: Pipeline for Sim-outorder……………………………………..…………….33
Figure 7.4: Architecture representing MIPS with FLWT unit………………………………34
Figure 7.5: Organization of FLWT unit …………….……………………………………35
Figure 7.6: Flowchart for calculating the total latency of an image data…………...……….36

Figure 8.1: Performance analysis of Liftpack with different picture sizes……….……….41
Figure 8.2: Performance comparison of filters with different polynomial degrees…………42
Figure 8.3: Comparison between performance of different filter types……….……….43
Figure 8.4: Comparison of hardware exec time in case1 and case……………………….…45

List of Tables

Table 5.1: Filter coefficients for N=2……………………………………………………….20
Table 5.2: Filter coefficients for N=4…………………………………………………..…20
Table 5.3: Lifting coefficients for $\tilde{N}$ =2 and L=16…………………………………………20
Table 5.4: Lifting coefficients for $\tilde{N}$ =4 and L=16…………………………………………20
Table 5.5: Extension for reversible transformation using 5-3 filter………………………...24
Table 5.6: Extension for reversible transformation using 7-9 filter……………...…………25

Table 8.1: Performance analysis of Liftpack with different picture dimensions……………41
Table 8.2: Performance analysis of Liftpack with different polynomial filters…...…..……42
Table 8.3: Performance analysis of wavelet transform using different kinds of filter……...43
Table 8.4: Simulation results for case 1 and case 2 using different picture dimensions……45

Chapter 1

Introduction

With the continuous expansion of the Internet data and multimedia applications, there is also an increased demand for high quality digital images. The subsequent increase in the image size is resulting an increase in the requirements of storage and transmission systems. In order to meet these requirements, data compression is developing as a popular topic in both the academic and industrial fields. The motivation behind image compression is that the image data sequence is very redundant and exhibits high correlation (a sample can easily be predicted from its neighboring sample values). Different algorithms are developed to discard these superfluous information, with least degradation of the quality and to represent the image in a compact form. One of the most widely used algorithms so far, is Discrete Cosine Transform (DCT), which uses Cosine functions of different frequencies to de-correlate the data.

The Discrete Wavelet Transform (DWT) was introduced in the field of image processing in 1980's and now it has emerged as a powerful tool for recent and future compression standards like MPEG-4 and JPEG 2000 standards. DWT exploits both spatial and frequency correlation of the data by dilations (or contractions) and translations of mother wavelet on the input data. It supports the multi resolution analysis of data, i.e., it can be applied to different scales according to the details required, which allows progressive transmission and zooming of the image without the need of extra storage. Another positive feature of wavelet transform is its symmetric nature, that is both the forward and the inverse transform have the same complexity, building fast compression as well as decompression routines. In comparison to DCT, DWT provides good signal to noise ratio and good visual quality [5][12].

The Lifting Scheme, proposed by Sweldens in 1995, is the fastest implementation of DWT algorithm using factorizing into lifting steps. It is easy to use in integer arithmetic without encountering problems due to precision or rounding. In addition, this scheme provides an easy inverse transform and will always result in a perfect reconstruction of the original picture, regardless of the precision of applied arithmetic.

1.1 Objective and Framework

Due to its advantageous features, the Wavelet Transform using Lifting Scheme is a very promising application in the area of multimedia applications, however, its purely software implementation has been a significant bottleneck. This thesis is aimed to accelerate Fast Lifting Wavelet Transform (FLWT) algorithm by implementing it on a hardware unit and analyze its performance with respect to the software implementation. The contributions described in this thesis can be summarized as follows:

- A new unit, co-existing with the functional units of a general-purpose MIPS processor, was designed for the Fast Lifting Wavelet Transform (FLWT). This unit simulates a realistic reconfigurable hardware model built in FPGA for the FLWT [17].The MIPS processor was assumed to operate in ordinary processor environments while the augmented FLWT unit would speed up the processing. The execution and the reconfiguration of the FLWT unit would be under the control of the “core” processor.

- A new instruction was also introduced as an ISA extension of MIPS architecture for the FLWT unit. This was inserted into the code so that the compiler would schedule FLWT operation to be executed in the added FLWT unit instead of general-purpose processor.

- The general-purpose MIPS processor simulator ‘Sim-outorder’ [2] was modified in order to introduce the FLWT unit. All the experiments were carried out on this modified simulator.

- As for the benchmarks, the Liftpack software [1] was modified and two standard FLWT algorithms from JPEG-2000 [14], namely the reversible wavelet transform (Le Gall 5-3) and the irreversible wavelet transform (Daubechies 9-7) were implemented. All of these algorithms are based on integer-to-integer wavelet transforms using lifting scheme on two-dimensional images but differ in filtering methods and boundary treatments of the data.

- The main performance metric considered during the analysis is the speed up of the FLWT operation, provided by the FLWT unit against the general-purpose processor. The results are analyzed for different benchmarks with different image dimensions (176*144, 352*288 and 720*560) ,by changing the filter parameters and with different memory models.

1.2 Organization of the Thesis

This report is divided into 9 chapters with an intention to provide adequate and sequential information about the thesis work. Chapter 2 provides overview of the image compression system and introduces the transform operations applied on images. Chapter 3 describes the theories behind wavelet transform and its application in image compression or decompression. Lifting scheme is introduced in Chapter 4 with the operational details including integer-to-integer transforms and boundary treatment. Chapter 5 describes the 3 different schemes in lifting wavelet transform, which is used as a benchmark in the thesis. Chapter 6 introduces the concept of reconfigurable computing and its application in wavelet transform. The experimental part of the thesis begins at Chapter 7 including the description of tool sets used and the methodology followed for the simulation. The simulation results and analysis on the results are explained in Chapter 8. The thesis ends with conclusions and suggestions for future research in Chapter 9.

Chapter 2

Image Compression

In this Chapter we will explain the digital representation of images are represented in computer systems and some image processing operations. Since the eventual application of our thesis work would be in image compression, this Chapter is included to give an overview of the image compression system and its components.

2.1 Digital Images and Compression

A digital color image comprises an two-dimensional (rectangular) array of pixels, which are the basic units used to represent the color information within the image. The color information can be represented using three values/components (e.g., RGB or YCrCb) according to the tri-chromatic theory. By combining all values belonging to one color component, a (so-called) gray image is obtained. Another widely used term is *resolution*, which defines the height and width of an image in pixels. In addition, resolution can also be used as a rough guide to the image quality. In general, higher resolution images provide more detail information resulting in higher image quality. However, this is at the expense of higher storage requirements.

A gray scale digital image can be considered as a rectangle where every point in the image has a luminance in the continuous range from black to white. These images contain a significant amount of *statistical redundancy*. That is, the samples are similar to each other so that one sample can be predicted fairly accurately from another. Similarly if the primary receiver of the multimedia signal is a human, which is mostly the case, there are also some redundant information which can not be perceived by the human system called *perceptual redundancy*. Compression attempts to remove these redundant information, resulting in the reduction of storage area and data rate requirements. With the reduced data rate requirements, the transmission costs are lowered and where a fixed transmission capacity is available, it provides a better quality of presentation. The storage requirements are significantly lowered and also increase the portability of multimedia programs.

2.2 Compression Techniques

Different compression techniques are developed in order to achieve higher compression and maintain high image quality. These compression methods can be divided broadly into two classes, namely loss-less and lossy compression.

Lossless compression guarantees that the original data can be reconstructed without any errors. This is important for sensitive applications like medical images where data loss is not acceptable. For picture data, loss-less compression is often used as the second step, after the lossy part.

Lossy compression does not allow an exact reconstruction of the original data, but higher compression rates can be achieved. Since the human visual system is not sensitive to some kinds of errors this method can be used in image data for high compression.

Lossy compression schemes are commonly based on transform coding. Such a coding scheme consists of three steps: transform coding, quantization and variable length encoding, as depicted in Figure 2.1.

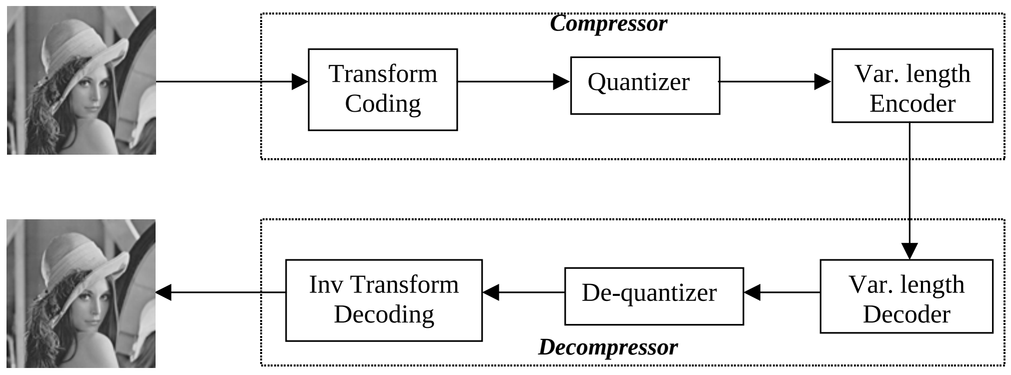

Figure 2.1: Block representation of digital image compression system

2.2.1 Transform Coding

Since an image is a continuous data sequence of gray values, the adjacent data are highly correlated. Transform methods are applied to de-correlate these data sequences so that statistical redundancy and irrelevant information are removed. The method involves predicting the data from its past or neighboring information.

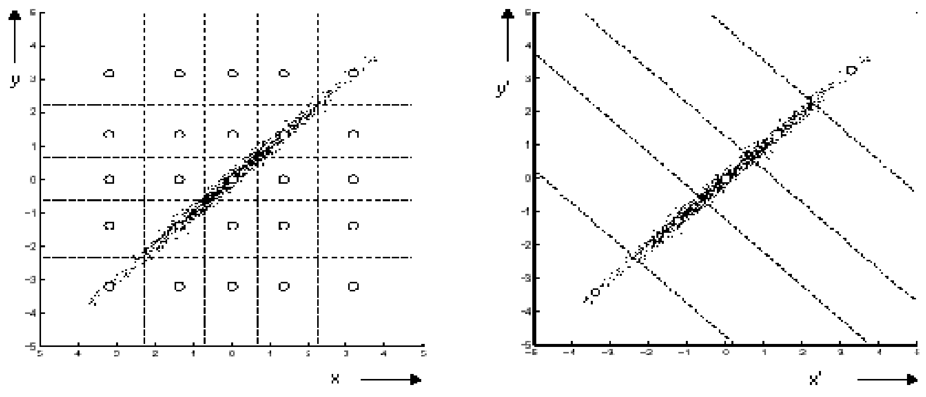

Figure 2.2: Transform coding: (left) corrolated image model with redundant reproduction coefficients (white dots), (right) de-correlation by rotating the co-ordinate axes

As illustrated in Figure 2.2, see for details [7], the distribution of the data elements are mainly confined to a specific region since they are highly correlated. So a better representation of the image, with less coefficients, is possible by *transforming* the axes of the co-ordinate system.

Some examples of transform techniques are Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT), Karhunen, Discrete Wavelet Transform (DWT) etc. In this thesis DWT has been implemented, which will be discussed in detail in Chapter 3.

2.2.2 Quantization

The transformed image is quantized to a finite number of pixels resulting a matrix of pixels. To obtain a finite precision representation the pixel values are restricted to a finite number of quantization levels. Continuous gray scale images are often quantized to 256 (8 bit) levels: black is 0, white by 255. Grey values between black and white are mapped into an integer number in the range 0 to 255. More quantization levels improve the quality of the quantized image, but at the cost of memory. This ends up with a discrete representation of the original gray scale image, consisting of a matrix of integer luminance values. As depicted in Figure 2.2, the white dots are the quantization output values.

2.2.3 Variable Length Encoding

The quantized values are further mapped into code-words, which enable a more efficient representation of the image data, in terms of size, for transmission and reconstruction of the image. Some popular coding schemes are: Entropy coding, Huffman coding, Variable length coding, Embedded Zero Tree algorithm etc.

Chapter 3

Wavelet Transform

In this Chapter, we introduce wavelets and the discrete wavelet transform from the classical viewpoint, based on the concept multi-resolution analysis. The Fast Wavelet Transform (FWT) and integer to integer transforms are also discussed because they allow the efficient calculation of a wavelet transform. We will not delve deeply into mathematical details, but we limit ourselves to the extent that is important for the remainder of this thesis. This Chapter will provide the theoretical background and put our work in context, representing where we began our work.

3.1 Wavelets

Wavelets, literally meaning small waves, are mathematical concepts for decomposing functions; say *f*, into sets of other functions known as wavelet bases $\psi_{a,b}(t)$.

$$f = \sum_{t} a_{a,b}\psi_{a,b}(t) \qquad \textit{(Eq. 3.1)}$$

To have an efficient representation of the signal *f* using only a few coefficients $a_{a,b}$, it is very important to use a suitable family of functions $\psi_{a,b}$. The functions $\psi_{a,b}$ should match the features of the data we want to represent. Real-world signals usually are limited in time (time-limited or space limited in case of pictures) and also limited in frequency (band-limited). Time-limited signals can be represented efficiently using a basis of block functions (Dirac delta functions for infinitesimal small blocks). But these block signals do not take care about frequency limitation. Band-limited signals can be represented efficiently using a Fourier basis like in DCT, but Sines and Cosines are not limited in time. Wavelets function as a compromise between the pure time-limited and band-limited basis functions, combining the best of both worlds.

In order to get the variable time-frequency localization ("*resolution*"), a wavelet called *mother wavelet* or *prototype* function $\psi(t)$ is defined as shown in Figure 3.1. Basis functions $\psi_{a,b}(t)$ are calculated as the scaled and translated version of the prototype.

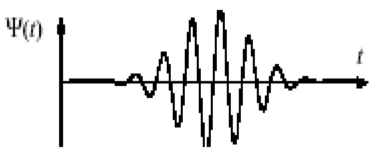

Figure3 .1: Wavelet prototype function

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-a}{b}) \qquad \textit{(Eq 3.2)}$$

Here, *b* is the scaling coefficient and *a* is the translating coefficient. Discrete Wavelet Transform (DWT) has the resolution level or scale *j* at time *k*.

$$\psi_{j,k}(t) = 2^{-\frac{j}{2}}\psi(2^{-j}t-k) \qquad \textit{(Eq 3.3)}$$

The scale functions are described in Figure3.2. Wavelets at different scales must form an (bi)orthogonal basis i.e. they must be mutually (bi)orthogonal.

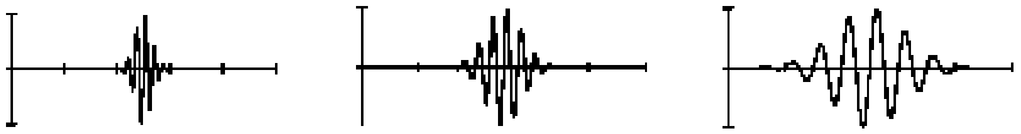
High frequencies
(large Δf small Δt)
mid frequencies
(medium Δf ,medium Δt)
low frequencies
(small Δf ,large Δt)

Figure 3.2: Illustration of wavelets at different scales

Considering *equatiosn 3.1 and 3.3*, signal can now be represented as the sum of a set of wavelet coefficients at infinite number of *scales*:

$$f(t) = \sum_{j,k} a_{j,k}\psi_{j,k}(t) \qquad \textit{(Eq 3.4)}$$

or $$a_{j,k} = \int_{-\infty}^{+\infty} f(t)\psi_{j,k}(t)dt \qquad \textit{(Eq 3.5)}$$

This equation resembles with the Fourier Transform Theorem and it is called the *classic wavelet transform* [3]

3.2 Multi Resolution Analysis

Another noteworthy feature of the wavelets is their property of *Multi Resolution Analysis* (MRA). MRA analyzes the signal at different frequencies with different resolutions. It is designed to give good time resolution and poor frequency resolution at high frequencies and good frequency resolution and poor time resolution at low frequencies

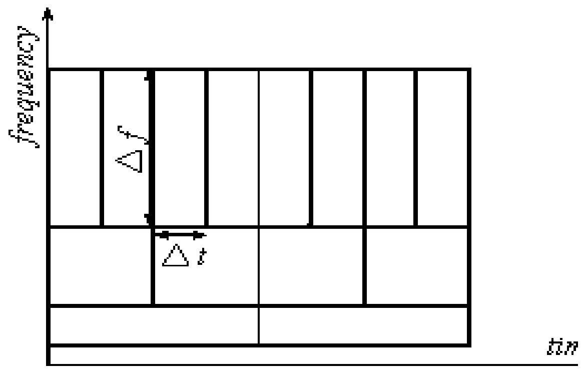

Figure 3.3:Multiresolution analysis on wavelet transform

The time-frequency behavior of Wavelet Transform can be described through Figure 3.3. Every box in Figure 3.3 corresponds to a value of the wavelet transform in the time-frequency plane. Each box represents an equal portion of the time-frequency plane by having equal area, but different proportions of time (Δ*t*) and frequency (Δ*f*). At low frequencies, the frequency resolution is better, since there is less ambiguity regarding the value of the exact frequency, but their time resolutions are poor since there is more ambiguity regarding the value of the exact time. At higher frequencies the time resolution gets better, and the frequency resolution gets poorer. In the case of image data, where the correlation between space and frequency is high, the large smooth areas can be represented by few wavelet coefficients (at low level of scale) and the detailed information, according to the degree of necessity, can be represented by the higher-level coefficients. This gives a very good data compression and excellent perceptual quality of the transformed picture, also with the perfect reconstruction capability. In short, wavelet transform of a signal is a multi-resolution representation of that signal where the wavelets are the basis functions, which at each resolution level de-correlate the signal.

3.3 Generating Wavelets Using Two-Channel Filterbanks

The wavelet transform coefficients can be generated using a 2 channel filterbanks called synthesis filters [10][11]. The generic form of 1-dimensional wavelet transform is shown in Figure 3.4. The input signal is split into two signals using a low-pass filter h(t) and its orthogonal or bi-orthogonal high-pass filter g(t). Then both signals are sub sampled by 2 (dropping every other sample) constituting one level of transform. Multiple levels or "scales" of the wavelet transform are made by repeating the filtering and decimation process on the low-pass branch outputs only. The process is typically carried out for a finite number resulting the *wavelet coefficients*.

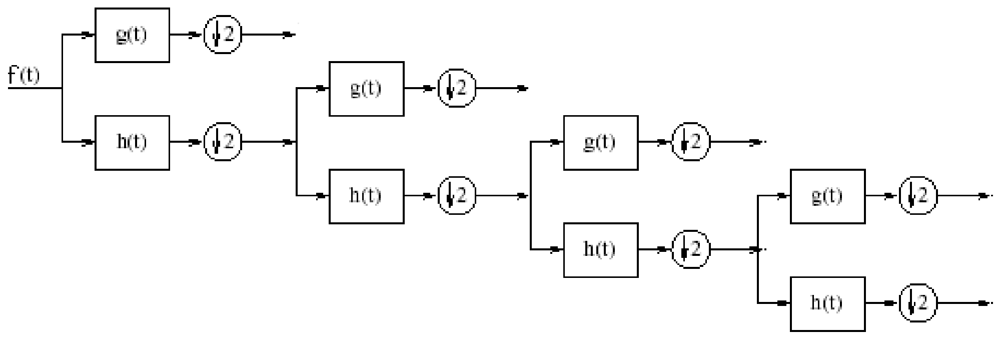

Figure 3.4: Iterated filter bank

The 1-D wavelet transform can be extended to a two-dimensional (2-D) wavelet transform using separable wavelet filters. With separable filters the 2-D transform can be calculated by applying 1-D transform to all the rows of the input and then repeating on all of the columns. An example of 1 level and 2 level 2-D transform with its corresponding notation is illustrated in Figure 3.5 and Figure 3.6 respectively. The example is repeated for 2-level transform (Figure 3.8) of the original "Lena" image (Figure 3.7).

Different filters that can be used to implement this scheme are: Quadrature Mirror Filter (QMF), Daubechies filters, Cubic B-splines etc, for the details see [7]. In this thesis, Daubechies 9-7 bi-orthogonal filter and Reversible 5-3 filter were implemented and worked with the bi-orthogonal wavelets. It is to be noted that the analysis filter and synthesis filter are transposes as well as inverse of each other; the whole filter bank is orthogonal. When they are inverses, but not necessarily transposes, they are bi-orthogonal [3].

3.4 The Fast Wavelet Transform

To approximate a sample by the wavelet function or vice versa, it requires applying a matrix whose order is equal to the number of sample points *n.* Since multiplication of *n* x *n* matrix by a vector costs the order of n^2 arithmetic operations, the computational intensity gets higher with increase in the number of sample points. However, the Discrete Wavelet Transform can be factorized into a product of a few sparse matrices using similarity properties. When these factors are applied to the multiplication with a vector, the order of operations reduces to *n*, thus called 'fast' [4].

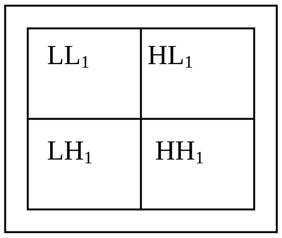

Figure3.5: The suband labeling scheme for one level, 2-D transform

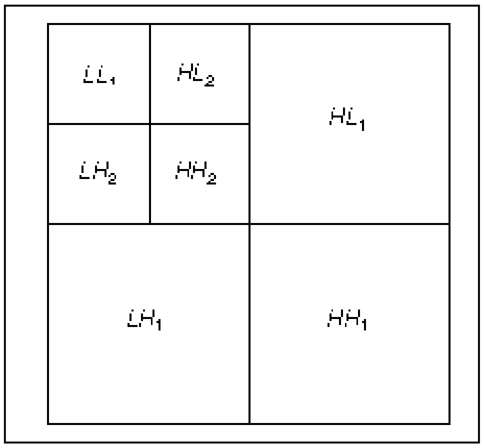

Figure 3.6: Labelling the 2 level, 2-D transform

Figure 3.7: Origninal Lena image

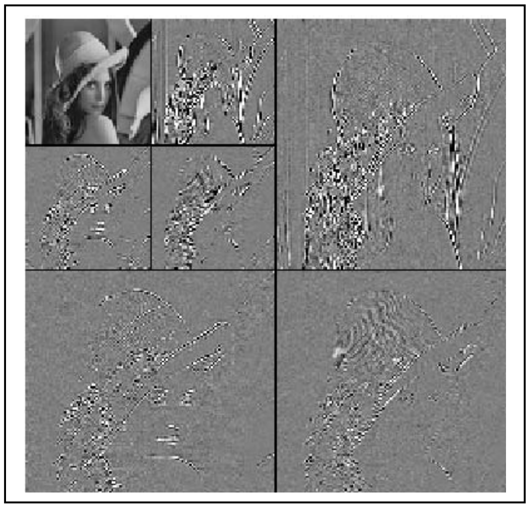

Figure 3.8: 2 level,2-D wavelet transform

3.5 Advantages of Using Wavelets

Here are some of the advantageous features of wavelet transforms that have made it popular for different applications:

1. One of the main features of wavelets that is their good de-correlating behavior which can be applied for efficient compaction of data.
2. Wavelets are localized in both the space/time and scale/frequency domains. Hence they can easily detect local features in a signal.
3. Wavelets are based on a multi-resolution analysis. A wavelet decomposition allows to analyze a signal at different resolution levels (scales).
4. Wavelets are smooth, which can be characterized by their number of vanishing moments. A function defined on the interval [*a*,*b*] has *n* vanishing moments if

   $$\int_a^b f(x)x^i dx = 0 \quad \text{for } i = 0,1,\ldots,n\text{-}1 \qquad (Eq.\ 3.6)$$

   The higher the number of vanishing moments, the more smooth the signals can be approximated in a wavelet basis.
5. The fast and stable (wavelets can be orthogonal or bi-orthogonal) algorithms to calculate the discrete wavelet transform and its inverse are already available [14][1].

After discussing the use of wavelets on different application, here are some facts due to which wavelets are becoming a popular tool for image compression. These are:

1. Wavelets provide good compression ratios especially for high compression ratios. Wavelets perform much better than competing technologies like DCT, both in terms of signal-to-noise ratio and visual quality. Unlike DCT, they show no blocking effect but allow for a graceful degradation of the whole image quality, while preserving the important details of the image.
2. The wavelet transforms are symmetric: both the forward and the inverse transform have the same complexity, allowing fast compression and decompression routines.
3. Multi-resolution allows progressive transmission and zooming, without the need for extra storage. For example we first transmit a thumbnail image, and gradually transmit and decompress more data to increase the resolution and overall image quality.
4. Wavelets are not only used for image compression, but also for various image-processing operations like de-noising the data, zooming and cropping the image. The possibility to combine image processing and compression is very appealing. However, image processing is difficult to carry out on the final encoded data stream: it must be done before the wavelet coefficients are quantized or encoded. But even then, the wavelet transform is equally important for both image processing and image compression.

Chapter 4

Lifting

In this Chapter, we introduce the lifting scheme, which is the implementation of classical wavelet transforms by a factorization in lifting steps [5] The main characteristics of this scheme includes its high speed, ease to operate on integer data and perfect reconstruction upon inverse transform. We worked out the details of integer wavelet transforms for a class of bi-orthogonal wavelets (Cohen-Daubechies-Feauveau) [18]

4.1 Lifting Scheme

Lifting scheme is an efficient implementation of a wavelet transform algorithm. All classical wavelet transforms can be implemented in the spatial domain by the factorization in lifting steps.

It was primarily developed as a method to improve wavelet transform, and then it was extended to a generic method to create so-called second-generation wavelets. The second-generation wavelets do not necessarily dilate and translate on one fixed function as the first generation wavelets (the wavelets produced by classic wavelet transform). They are much more flexible and can be used to define a wavelet basis on an unequal interval or on an irregular grid, or even on a sphere while retaining the powerful properties of first generation wavelets like fast transform, localization and good approximation. In this thesis we work out the details of second-generation wavelet transforms for bi-orthogonal wavelets, a technique first developed by Cohen, Daubechies and Feauveau [18].

The lifting scheme is an implementation of the filtering operations at each level. This process de-correlates the data resulting a compact representation. The algorithm can be described in three phases, namely: Split phase, Predict Phase and Update Phase, as illustrated in Figure 4.1. Assume the scheme starts at data set of $\lambda_{0,k}$ where k represents the data element and zero signifies the original data level.

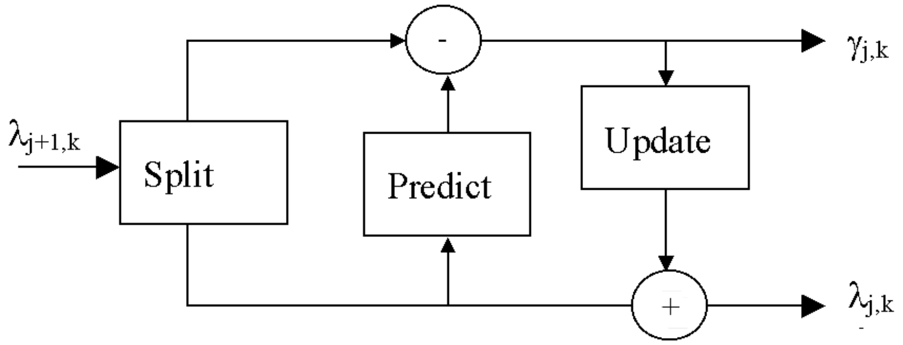

Figure 4.1: The Lifting Scheme: Split, Predict and Update

4.1.1 Split Phase

In the first stage the data set $\lambda_{0,k}$ is split into two other sets (see Figure 4.1): the $\lambda_{-1,k}$ and the $\gamma_{-1,k}$. The negative indices have been used according to the convention that the smaller the data set, the smaller the index. The new data at level 1 corresponding to the data at level 0 are given as:

$$\lambda_{-1,k} = \lambda_{0,2k} \qquad (Eq.\ 4.1)$$

$$\gamma_{-1,k} = \lambda_{0,2k+1} \qquad (Eq.\ 4.2)$$

The splitting is done by separating the set of even samples and the odd samples. This is also referred to as the *lazy wavelet transform* because it doesn't de-correlate the data but just sub-samples the signal into even and odd samples.

4.1.2 Predict Phase (Dual Lifting)

The next step is to use $\lambda_{-1,k}$ subset to predict $\gamma_{-1,k}$ subset with the use of prediction function *P*, see Figure 4.1 and 4.4, independent of the data, so that

$$\gamma_{-1,k} := P(\lambda_{-1,k}) \qquad (Eq\ 4.3)$$

The more the correlation present in the original data, the closer will be the predicted value with the original $\lambda_{-1,k}$. Now, the set $\gamma_{-1,k}$ will be replaced by the difference between itself and its predicted value $P(\lambda_{-1,k})$. Thus,

$$\gamma_{-1,k} := \lambda_{0,2k+1} - P(\lambda_{-1,k}) \qquad (Eq\ 4.4)$$

Different functions can be used for prediction of odd samples. The easiest choice is to predict that an odd sample is just equal to its neighboring even sample. This prediction method is result to the Haar wavelet. Obviously this is an easy but not realistic choice, as there is no reason why the odd samples should be the equal to the even ones. The prediction functions can be broadly divided into *piecewise linear* (of order 2) and non-linear or *interpolating subdivision*.

In the ***piecewise linear*** model depicted in Figure 4.2, the odd samples $\lambda_{0,2k+1}$ are predicted as the average of its two (even) neighbors, $\lambda_{-1,k}$ and $\lambda_{0,k+1}$, which is given by:

$$\gamma_{-1,k} := \lambda_{0,2k+1} - \frac{1}{2}(\lambda_{-1,k} + \lambda_{0,k+1}) \qquad (Eq\ 4.5)$$

In this model *P* is a *piecewise linear* over intervals of length 2. If the original signal matches with the model, all wavelet coefficients ($\gamma_{-1,k}, \forall\ k$) will be zero. In other words, the wavelet coefficients measure to which extent the original signal fails to be linear. In terms of frequency content, the wavelet coefficients capture the high frequencies present in the original signal.

The ***interpolating subdivision*** method gives better approximation of the higher order signals. This subdivision model uses the same idea of piecewise linear but uses 2 or more neighbors to either side for predicting. Depending on the order of subdivision (interpolation), denoted by *N*, these wavelet coefficients can measure failure to predict. For instance, *N* =2 gives the indication of the extent of failure to be linear and *N*=4 measures the failure to be cubic. It can be seen that *N* is important because it sets the smoothness of the interpolating function used to find the wavelet coefficients (high frequencies). This function is referred to as the dual wavelet and to *N* as the number of dual vanishing moments. Thus the number of *dual vanishing moments* defines the degree of the polynomials that can be predicted by the dual wavelet. Figure 4.3 illustrates linear interpolation (*N*=2), where one even sample at each side is used for predicting the odd sample, while Figure 4.4 depicts cubic interpolation (*N*=4), where two even samples at each side are used for prediction.

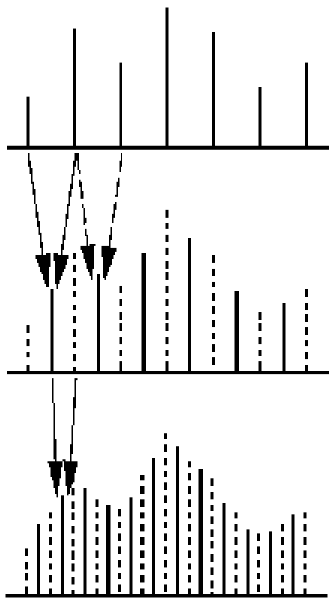

Figure 4.2: linear interpolation, N=2

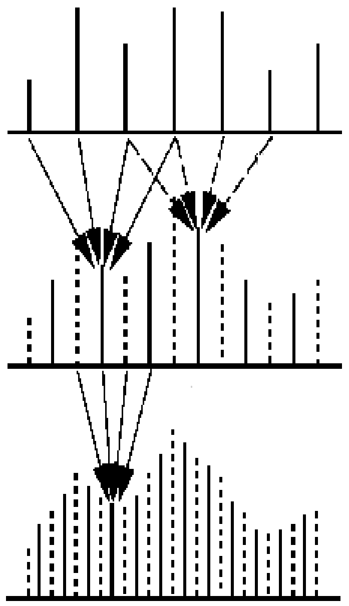

Figure 4.3: cubic interpolation, N=4

The best value of *N* (always even) corresponds to the polynomial degree *N*-1of the original signal, which results the predicted value equal to the original value. This will lead the transform coefficients to have maximum number of zero and maximum compression.

With a good prediction, a very compact version of the original data set can be obtained by iterating this scheme, say *n* times. But in the cases of image compression, it is also necessary to maintain some global properties of original data set in lower levels of prediction, for instance, same overall brightness (i.e. the same average pixel value). If we iterate the prediction scheme in the image data till there is only one pixel, it will be an arbitrary pixel from the original image, badly affected by aliasing. So solve this problem, a new scheme with a new operator is applied to the predicted data.

$$\begin{tikzcd}
\cdots & \lambda_{j,2k} \arrow[dl, "-1/2"'] \arrow[d] \arrow[dr, "-1/2"] & \lambda_{j,2k+1} \arrow[d] & \lambda_{j,2k+2} \arrow[dl, "-1/2"'] \arrow[d] \arrow[dr, "-1/2"] & \cdots \\
\cdots \arrow[dr, "1/4"] & \lambda_{j,2k} \arrow[d] & \gamma_{j-1,k} \arrow[dl, "1/4"'] \arrow[d] \arrow[dr, "1/4"] & \lambda_{j,2k+2} \arrow[d] & \cdots \arrow[dl, "1/4"'] \\
\cdots & \lambda_{j-1,k} & \gamma_{j-1,k} & \lambda_{j-1,k+1} & \cdots
\end{tikzcd}$$

Figure 4.4: The lifting scheme: Predict $\gamma_{-1,k}$ *and update* $\lambda_{-1,k}$

4.1.3 Update Phase (Primal Lifting)

In this stage the coefficient $\lambda_{-1,k}$ is lifted with the help of the neighboring wavelet coefficients so that a certain scalar quantity *Q*, like for example the mean is preserved.

$$Q(\lambda_{-1,k}) = Q(\lambda_{0,k})$$

For this reason a new operator *U* is applied update $\lambda_{-1,k}$, see figure 3.4.

$$\lambda_{-1,k} := \lambda_{-1,k} + U(\gamma_{-1,k}) \qquad (Eq\ 4.6)$$

In this phase, a *scaling function* is calculated from the previously calculated wavelet coefficients to maintain some properties among all the λ coefficients throughout every levels. This will create a *real wavelet* that will maintain some desired properties from the original signal. The order of this of this function will be some even value of $\tilde{N}$ called *real vanishing moment*, not necessary equal to N. The basic properties that are to be preserved by the wavelet function are the *moments*. In other words, the transform preserves $\tilde{N}$ moments of the each level of λ.

The number of data operated in each level is given in Figure 4.5

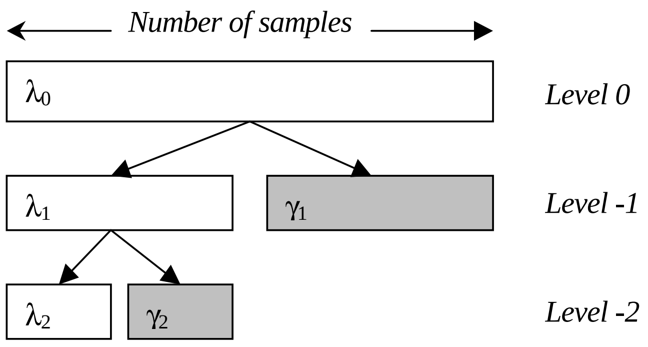

Figure 4.5: Number of samples in different levels

4.2 Inverse Transform

The inverse transform for lifting scheme is very clear and trivial. It is just the reverse data flow in the setup of forward transform with small changes like switching additions and subtractions and also switching divisions and multiplications. Hence, the algorithm for inverse transform becomes as depicted in Figure 4.6:

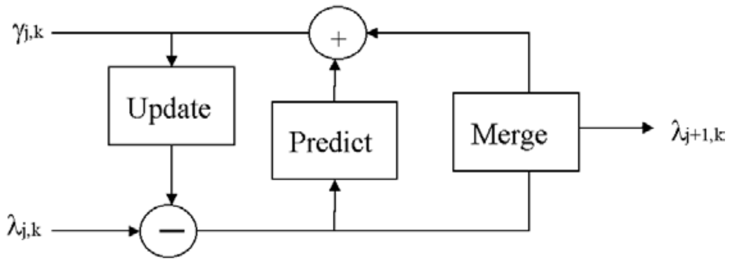

Figure 4.6: The Lifting Scheme, inverse transform: Update, Predict and Merge

The sequence of operations to carry out the inverse lifting wavelet transform can be represented according to the following equations:

1- Update phase $\lambda_{j,k} = \lambda_{j,k} - U(\gamma_{j,k})$

2- Predict phase $\gamma_{j,k} = \gamma_{j,k} + P(\lambda_{j,k})$

3- Merge phase: $\lambda_{j+1,2k} = \lambda_{j,k}$
$\lambda_{j+1,2k+1} = \gamma_{j,k}$

4.3 Integer-to-Integer Transform

In many applications, especially in image processing, the input data consist of integer samples. But all of the wavelet coefficients are floating point values even though the input samples are integers because filter coefficients used in transform filters are mostly rational or real numbers[12]. Integer data is very important and useful for fast implementation of the discrete wavelet transform, particularly in hardware, due to its efficient storage and encoding.

The Lifting Scheme can be easily modified to map integers-to-integers, by adding scaling and rounding operations, at the expense of introducing a non-linearity in the transform. Such integer transforms are reversible, disregarding the quantization and encoding non-linearities.

First, the Predict and Update filter coefficients are scaled and rounded to integers. Now integer arithmetic can be used, as both the data and the filter coefficients are integer values. Rounding of the filter coefficients introduces some error, denoted as *E* below.

Calculated predict and update operations are denoted inside the curly brackets.

Forward transform:

$$\gamma_{j,k,forward,} = \gamma_{j,k,original} - \{P(\lambda_{j,k}) + E\}$$
$$\lambda_{j,k,forward} = \lambda_{j,k,original} + \{U(\gamma_{j,k}) + E\}$$

The introduced error *E* is fully deterministic, i.e. while calculating the inverse transform; we introduce exactly the same error as we did while calculating the forward transform. Equations below show that irrelevant of the rounding error, the original values are exactly reproducible; which means a perfect reconstruction of the original image.

Inverse transform:

$$\begin{aligned}\gamma_{j,k} &= \gamma_{j,k,forward} + \{P(\lambda_{j,k}) + E\}\\ &= \gamma_{j,k,original} - \{P(\lambda_{j,k}) + E\} + \{P(\lambda_{j,k}) + E\}\\ &= \gamma_{j,k,original}\end{aligned}$$

$$\begin{aligned}\lambda_{j,k} &= \lambda_{j,k,forward} - \{U(\gamma_{j,k}) + E\}\\ &= \lambda_{j,k,original} + \{U(\gamma_{j,k}) + E\} - \{U(\gamma_{j,k}) + E\}\\ &= \lambda_{j,k,original}\end{aligned}$$

4.4 Advantages of the Lifting Scheme

Summarized, the lifting scheme has the following immediate advantages, when compared to the classical filter bank algorithm:

1. Lifting leads to a speed-up when compared to the classic implementation of wavelet transforms. The complexity of wavelet transform is in the order of *n* (equal to the number of samples), which is more efficient than FFT with its complexity in the order of *nlogn*. Lifting speeds up with another factor of two for long filters.[4] Hence it is also referred as *fast lifting wavelet transform* (FLWT).
2. All operations within lifting step can be done entirely parallel while the only sequential part is the order of lifting operations.
3. Lifting allows for adaptive wavelet transforms. This means the analysis of a function can start from the coarsest level and then build finer levels by refining in the areas of interest.
4. Lifting can be done in-place, which means auxiliary memory is not needed (see Figure 4.4) because it does not need samples other than the output of the previous lifting step. The old stream in replaced by the new one at every summation point.
5. It is easy to build non-linear wavelet transform by Lifting, for example, integer-to-integer transform. Such transforms are important for hardware implementation and loss-less image coding.

4.5 Boundary Treatment

Real-world signals do not extend infinitely in time or space, but are limited to a finite interval (Figure 4.7). However, filter bank algorithms assume infinite signal lengths because the filters depend on the past and/or future sample values. So when the signal comes close enough to the edge, the filters need some sample values that are not defined. There are two general approaches that can be used to address this problem. The first approach is to extend the data so that the sample is defined for all the indices, called *extension method*. The second approach is to change the filters at the boundaries called *boundary filtering*. In the following paragraphs these approaches are discussed.

4.5.1 Classical Extension Methods

If we employ periodization or zero padding in order to avoid the boundary discontinuity, the transform will result large coefficients. This will induce artifacts in the image and a severe encoding inefficiency. Therefore, as a solution to this problem, classical signal processing extend the data for the computations near the border, by replicating the signal using one of the following methods:

Periodic Extension

The finite signal is extended periodically by putting copies of itself in front of and behind the original signal (Figure 4.8). After the wavelet transform, the coefficients that lie outside of the interval of defined signal is simply discarded. These discarded coefficients can be recovered easily because they are same as the retained coefficients. However, unless the first and the last sample have the same value, we introduce unwanted discontinuities at the boundaries of the original signal. These discontinuities will locally enlarge the wavelet coefficients and make compression of the signal more difficult.

Symmetric Extension

A relatively easy solution for handling the finite length signals is to extend them such that they become symmetric and periodic (Figure 4.9). There are many ways to symmetrically extend the signal. The main difference between these ways lies in the number of times the first and last samples are repeated in the extended signal. In this thesis, only (1,1) symmetric extension was used,(see Chapter 5) such that, the first and last samples of the signal appear once (they are not repeated), as shown in Figure 4.10. After filtering the extended signal, the number of coefficient becomes double than the original signal but half of them can be discarded if the filters are symmetric, yielding the same number of coefficients as the original signal length.

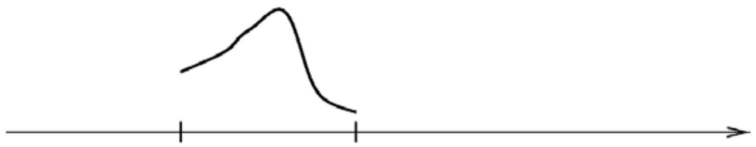
Figure 4.7: A finite signal

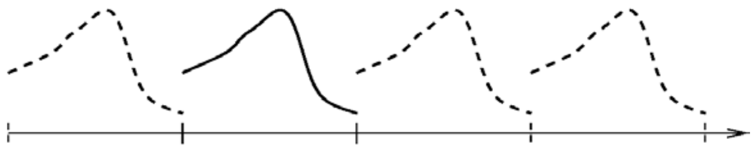
Figure 4.8: Periodic extension of signal

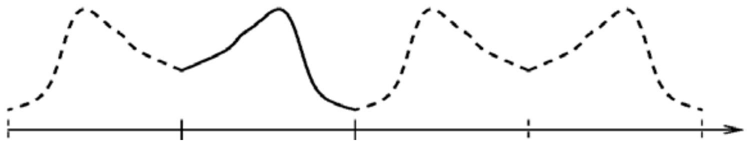
Figure 4.9: Symmetric extension of signal

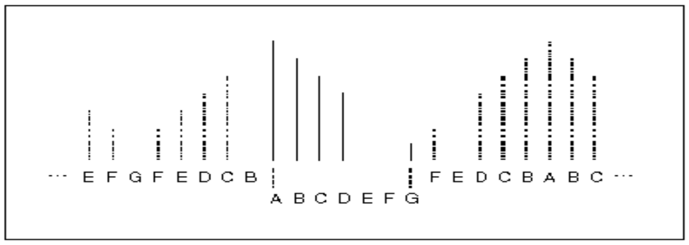

Figure 4.9: Periodic symmetric extension (1,1) of the finite length signal "ABCDEFG."

Signal Extension with Lifting

One of the positive features of lifting scheme is that the symmetrical extension is always possible even if the wavelets are orthogonal or filter banks are non-symmetric [12]. The signals are extended symmetrically, as shown in Figure 4.10, in each lifting step. The extension parameters are defined depending on the first and last sample being "even" or "odd'. The possible four cases, as presented graphically in Figure 4.10, are:

1. **FELO** First even, last odd
2. **FELE** First even, last even
3. **FOLO** First odd, last odd
4. **FOLE** First odd, last even.

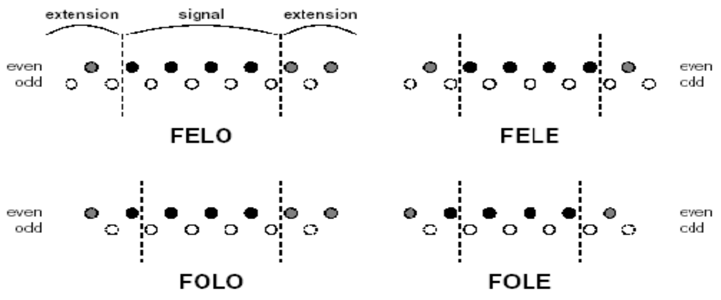

Figure 4.10: Cases of symmetrical extension using lifting

4.5.2 Boundary Filtering

One of the sophisticated approaches to deal with the finite length signals is to apply boundary filter banks. In this approach the filter structure or/and filters at the boundaries are changed for time varying filtering. More detailed implementation of this process is discussed in Chapter 5.

Chapter 5

Software Implementation of Selective Lifting Schemes

This Chapter describes three different lifting schemes that were implemented as benchmarks for this thesis. These schemes are namely; modified *LIFTPACK*, *Reversible Wavelet Lifting Transform and Irreversible Wavelet Lifting Transform.* Liftpack is a available software by [1] and it was modified in order to employ it as integer to integer transform. The Reversible and Irreversible Wavelet Transforms are the standards of JPEG-2000 for integer to integer transform and we implemented them in software using C. In order to make a comparative study among these schemes, in this Chapter emphasis has been placed on the points of their differences in filtering algorithm, boundary treatment and lifting algorithm.

5.1 Liftpack

Liftpack is a software package with object-oriented modules written in C for fast calculation of 2-D bi-orthogonal wavelet transforms using the lifting scheme [1]. . We intended to employ Lifpack as a benchmark of lifting wavelet transform in software and in hardware reconfigurable unit. The original version of Liftpack uses floating point operations since the filter and lifting coefficients are in fractional numbers. However, the floating point operations are very costly for hardware unit. Therefore all the filter and lifting coefficients were converted to integer format by scaling and rounding.

5.1.1 Filter Implementation

The filtering operations of the Liftpack are based on polynomial subdivision or interpolation scheme as described in Section 4.1.2.The prediction function *P* uses the polynomial interpolation of order *N*-1 to find the predicted values. The higher the order of this function, the better would be the approximation of γ coefficients from the λ coefficients, provided the correlation between the coefficients. The interpolating polynomials for the predict stage are called *filter coefficients* and for the update stage they are called *lifting coefficients*.

An algorithm called *Neville's algorithm* is used that generates the interpolating polynomials for predict stage (filter coefficients), for details see [1]. Both the filter coefficients and lifting coefficients are stored in matrices. For the filter coefficients, the matrix consists of N/2 + 1 rows of each N columns, so in total it has (N/2 + 1)*N entries, where N is the number of dual vanishing moments. Table 5.1 shows the filter coefficients for N=2, while Table 5. *2* illustrates the filter coefficients in case N=4. Notice that these tables contain N+1 rows; one row for each interpolating case. Notice also the symmetry in the in the rows. This is why we can store them in N/2 +1 rows.

Cases		Coefficients			
# λ's on left	# λ's on right	$k-3$	$k-1$	$k+1$	$k+3$
0	2			-0.5	1.5
1	1		0.5	0.5	
2	0	1.5	-0.5		

Table 5.1: Filter coefficients for N=2

Cases		Coefficients							
# λ's on left	# λ's on right	$k-7$	$k-5$	$k-3$	$k-1$	$k+1$	$k+3$	$k+5$	$k+7$
0	4					2.1875	-2.1875	1.3125	-0.3125
1	3				0.3125	0.9375	-0.3125	0.0625	
2	2			-0.0625	0.5625	0.5625	-0.0625		
3	1		0.0625	-0.3125	0.9375	0.3125			
4	0	-0.3125	1.3125	-2.1875	2.1875				

Table 5. 2: Filter coefficients for N=4

For lifting coefficients, the matrix consists of $n*G*\tilde{N}$ rows, where $\tilde{N}$ is the number of real vanishing moments, *n* is the maximum number of iterations and *G* is the number of γ coefficients. Thus a separate set of $\tilde{N}$ coefficients is used for each γ to update $\tilde{N}$ λs. In additions, different iteration levels use different sets of lifting coefficients. Tables 5.3 and 5.4 depict the lifting coefficients for $\tilde{N}$ =2 and $\tilde{N}$ =4, respectively. In both cases the signal length is 16 (L=16).

	Moments ($\tilde{N}=2$)							
Level	γ_1	γ_2	γ_3	γ_4	γ_5	γ_6	γ_7	γ_8
1	(0.4,0.2)	(0.25,0.25)	(0.25,0.25)	(0.25,0.25)	(0.25,0.25)	$(0,0.\bar{6})$	$(0.2\bar{6},0.2)$	$(-0.1\bar{3},0.4)$
2	$(0.5\bar{3},0.1\bar{6})$	(-4.5,8)	$(0.\overline{27},0.1883)$	$(-0.\overline{18},0.4935)$				
3	(0.5588,0.2294)	(-0.4118,0.4941)						

Table 5.3: Lifting coefficients for $\tilde{N}$ =2 and L=16

Level 1

	Integral	1st. Moment	2nd. Moment	3rd. Moment
γ_1	0.184628	0.387125	-0.131771	0.0272476
γ_2	-0.105268	0.295406	0.268679	-0.0284388
γ_3	0.0079594	0.32098	0.190416	0.014943
γ_4	-1.12943	1.86682	-0.417554	-0.0467274
γ_5	-0.0431522	0.360257	0.145585	0.0507862
γ_6	-0.0377447	0.309029	0.233008	0.0178615
γ_7	0.0159595	-0.0828344	0.311458	0.333931
γ_8	-0.0180709	0.0783563	-0.121172	0.239113

Level 2

	Integral	1st. Moment	2nd. Moment	3rd. Moment
γ_1	0.55218	0.30749	0.0129941	-0.0883121
γ_2	-0.179825	0.327006	0.236753	-0.00309677
γ_3	-0.0683047	0.0619364	0.154456	0.34
γ_4	-0.183823	0.121297	-0.186652	0.23924

Table 5.4: Lifting coefficients for $\tilde{N}$ =4 and L=16

In this thesis , in order to make all the transform operations in integer format, all these filter coefficients were scaled and rounded for the best integer approximation. For example:

$$filter\ coefficients = \left\lfloor (filter\ coefficients * scale\_predict) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.1)}$$

$$lifting\ coefficients = \left\lfloor (lifting\ coefficients * scale\_update) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.2)}$$

5.1.2 Boundary Treatment

The interpolation scheme allows us to easily accommodate interval boundaries for finite sequences, using the variable filter coefficients. Here we explain this concept with an example of cubic interpolating sub-divison (*N=4*), described in Section 4.1.2. Each of the cases mentioned below are encountered during the 1-D transform of the image, see Figure 5.1, and they require different sets of filter coefficients.

Case 1*: Near Left Boundary:* More λ coefficients on the right side of the γ coefficient than on the left side.

- 1 λ on the left and 3 λs on the right (due to the splitting, we always have a λ in the first position)

Case 2: *Middle*: Enough λ coefficients on either side of the γ coefficient.

- 2 λs on the left and 2 γs on the right

Case 3: *Near Right Boundary*: More λ coefficients on the left side of the γ coefficient than on the right side_

- 3 λ s on the left and 1 λ on the right
- *4* λ s on the left and 0 λ s on the right

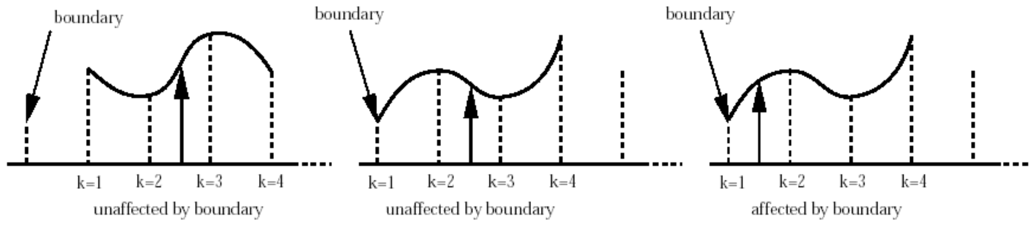

Figure 5.1: Behavior of the cubic interpolating subdivision near the boundary

Due to symmetry [18] all the cases on the right side are the opposite to the cases on the left side. For example, the coefficients used for the case "3 λ s on the left and 1 λ on the right" are the same as the ones used for the case "1 λ on the left and 3 λs on the right", but in opposite order. Thus the total cases will be *N*/2+1, one for the middle case and N/2 for the symmetric boundary case. Thus, using interpolation scheme and Neville's algorithm, *N*/2+1 sets of filter coefficients each with *N* coefficients are generated that will help us to find the correct approximation.

A similar story holds for the update phase; near the boundaries, when there are not enough λs, we use the set of $\tilde{N}$ λs near the edge. However, the lifting filter (update filter) contains a separate set of coefficients for *each* γ, thus also a different set of coefficients for each not-affected λs.].

5.1.3 Transform Algorithm

Using filter coefficients and lifting coefficients, the family of bi-orthogonal wavelets can be generated with special boundary treatments. Different sets of filter coefficients are used to find the transform of data near the boundaries and the rest of the data. In short the sequence of 1-D fast lifted wavelet operation are represented by:

Predict phase: $$\gamma_{j,k} = \gamma_{j,k} + \sum_{\tilde{N}} \lambda_{j,k} * \textit{filter coefficient} \qquad \textit{(Eq 5.3)}$$

where, $k = 0,1,2 \ldots N$

Since the *filter coefficient* are scaled up for integer approximation, the final λ is calculated by

$$\gamma_{j,k} = \gamma_{j,k} / scale\_predict \qquad \textit{(Eq 5.4)}$$

Update phase: $$\lambda_{j,k} = \lambda_{j,k} + \sum_{\tilde{N}} \gamma_{j,k} * \textit{lifting coefficient} \qquad (\textit{Eq 5.5})$$

$k = 0,1,2 \ldots \tilde{N}$

The final value of γ is calculated by;

$$\lambda_{j,k} = \lambda_{j,k} / scale\_update \qquad \textit{(Eq 5.6)}$$

This sequence of operation is performed on the signal using "*square 2-D*" method. In this method the 1-D transform is applied first to all the rows and then to all the columns. This is done at every level to create square window transform, which gives better frequency transform than rectangular window transform.

The total number of iteration "*n*" in this transform depends upon the 3 factors, which are: the signal length (L), the number of dual vanishing moments (*N*) and, the number of real vanishing moments ($\tilde{N}$),

$$n = \left\lfloor \log_2 \left(\frac{L-1}{N_{\max} - 1} \right) \right\rfloor \qquad (\textit{Eq 5.7})$$

where $N_{max} = \max(N, \tilde{N})$. The signal not necessarily have to have dyadic dimension, interpolating subdivision method uses the right treatment of the boundaries for every case.

Similarly, the inverse transform can be derived very easily from the forward transform just by reversing the operations and toggling + and -. This leads to the following algorithm:

Update phase: $$\lambda_{j,k} = \lambda_{j,k} - \sum_{\tilde{N}} \gamma_{j,k} * \textit{lifting coefficient} \qquad (\textit{Eq 5.8})$$

$$\lambda_{j,k} = \lambda_{j,k} / scale\_update \qquad \textit{(Eq 5.9)}$$

Predict phase: $$\gamma_{j,k} = \gamma_{j,k} - \sum_{N} \lambda_{j,k} * \textit{filter coefficient} \qquad (\textit{Eq 5.10})$$

$$\gamma_{j,k} = \gamma_{j,k} / scale\_predict \qquad \textit{(Eq 5.11)}$$

5.1.4 Modifications on Liftpack Source Code

In order to use Liftpack as a benchmark for the lifting wavelet transform, the option employed in this thesis, from the original version of the Liftpack [1], was ‘polynomial filters with even numbers of vanishing moments’. This project dealt with 2-D bi-orthogonal wavelet transforms using the lifting scheme, in-place format, interfacing through command-line parser. The PGM format was used as the representation of picture and TXT format as the output of FLWT encoder.

The following are the marked modifications, made in LIFTPACK:

- Since floating point operation are costly to be implemented in the reconfigurable unit, Liftpack was converted to integer-to-integer wavelet transform. The floating-point is introduced in the transform operation from filter coefficients. So first of all, the filter coefficients were scaled up and rounded to get the best approximation as explained in equation 5.1.

- Liftpack uses dynamic memory allocations for its data structures of arbitrary size. To make it useful for the placement in reconfigurable unit, we modified it work on static memories,

- In Liftpack, the filter coefficients are calculated as a part of the FLWT operation inside the software. Since it is calculated only once for the whole operation, in this thesis the filter coefficients were used as an input parameter. Hence Liftpack spends no time on calculating the filter coefficients rather they are directly taken from memory for the FLWT operation.

5.2 Reversible Wavelet Transform

This transform is employed by JPEG-2000 standard [14] as lifting based implementation of filtering by 5-3 filter, see also [9]. All the operations are changed into integer based in order to map it into the reconfigurable unit. Before proceeding further into the lifting algorithm here are the boundary treatments and filtering used by the transform:

5.2.1 Boundary Treatment

The boundary effects are solved by using periodic symmetric extension of the signal (Section 4.5.1) prescribed as ‘*1D_EXTR Procedure’* by JPEG 2000, see [14] At lower levels of decomposition, the number of samples to be extended can exceed the signal length. In such cases the extensions are carried out in the sequence as shown in Figure 5.2.

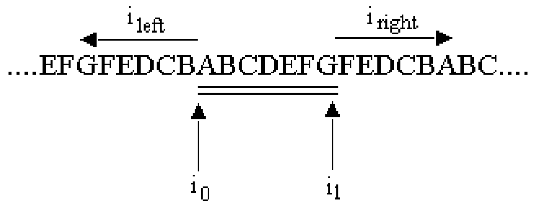

Figure 5.2: Periodic symmetric extension of signal

The minimum but sufficiently large values of the extension parameters i_{left} and i_{right} are given in the Table 5.5. The method to find i_{left} and i_{right} are discussed in Section 4.4 and Figure 4.10.

i_o	i_{left}	i_{right}
even	2	1
odd	1	2

Table 5.5: Extension for reversible transformation using 5-3 filter

5.2.2 Filter Procedure

This transform use linear phase bi-orthogonal filters with symmetric inputs and outputs called Le Gall or 5-3 filter [6] As the name suggests, it has 5 taps as analysis filter and 3 taps as synthesis filter. The filtering process is carried out on the extended signal for which alternately, odd coefficients values of the signal are updated with a weighted sum of even coefficient values and even coefficient values are updated with a weighted sum of odd coefficient values. The filter coefficients are calculated to be $\frac{1}{2}$ and $\frac{1}{4}$ for predict and update phase respectively .Since these fractional number in filter coefficient introduces floating point operations, which will be very costly for the hardware implementation, we scaled up the coefficient and rounded to the integer number.

$$filter\_predict = \left\lfloor \frac{1}{2} * scale + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.12)}$$

$$filter\_update = \left\lfloor \frac{1}{4} * scale + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.13)}$$

5.2.3 Transform Algorithm

The reversible transform algorithm using 5-3 is a integer to integer wavelet transform where after every filtering operations the coefficients are rounded to integer value. Following the same set of notations as mentioned in Section 4.1, the reversible FLWT transform can be represented as the sequence of operations on low-pass signal $\lambda_{0,k}$ and the high-pass signal $\gamma_{0,k}$. The final λ and γ coefficients are determined by scaling down the calculated coefficients from predict and update phase because the filter coefficients were scaled up in equation 5.3 and 5.4.

Predict phase:

$$\gamma_{-1,k} = \gamma_{0,k} + \left\lfloor filter\_predict * (\lambda_{0,k} + \lambda_{0,k+1}) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.14)}$$

$$\gamma_{-1,k} = \frac{\gamma_{-1,k}}{scale} \qquad \textit{(Eq 5.15)}$$

Update phase:

$$\lambda_{-1,k} = \lambda_{0,k} + \left\lfloor filter\_update * (\gamma_{-1,k-1} + \gamma_{-1,k}) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.16)}$$

$$\lambda_{-1,k} = \frac{\lambda_{-1,k}}{scale} \qquad \textit{(Eq 5.17)}$$

All the wavelet transforms described in JPEG-2000 codec are fundamentally one dimensional (1-D) in nature. In this thesis, two-dimensional (2-D) transforms were formed by applying 1-D transforms in horizontal direction (rows) and then in vertical directions(columns). The number of iterations to be performed on the data in order to get the optimum compression, while retaining some of its basic properties, is given by

$$n = \lfloor \log_2 (L-1) \rfloor \qquad \textit{(Eq 5.18)}$$

Similarly, the inverse transform of reversible FLWT is very trivial and results perfect reconstruction. The sequence of operations are given as:

Update phase:

$$\lambda_{-1,k} = \lambda_{0,k} - \left\lfloor filter\_update * (\gamma_{-1,k-1} + \gamma_{-1,k}) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.19)}$$

$$\lambda_{-1,k} = \frac{\lambda_{-1,k}}{scale} \qquad \textit{(Eq 5.20)}$$

Predict phase:

$$\gamma_{-1,k} = \gamma_{0,k} - \left\lfloor filter\_predict * (\lambda_{0,k} + \lambda_{0,k+1}) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.21)}$$

$$\gamma_{-1,k} = \frac{\gamma_{-1,k}}{scale} \qquad \textit{(Eq 5.22)}$$

5.3 Irreversible Wavelet Transform

This transform method is prescribed by JPEG-2000 standard [14] for irreversible wavelet transform by using Daubechies 9-7 filter pair [5][9]. The analysis filter has 9 coefficients and synthesis filter has 7 coefficients. The algorithm was implemented all by integer operations.

5.3.1 Boundary treatment

This transform uses the same periodic symmetric extension for boundary treatment *'1D_EXTR Procedure'* as explained in the case of reversible transform (Section 5.2.1 and Figure 5.2) except for the extension parameters. The minimum but sufficiently large values of the extension parameters i_{left} and i_{right} for irreversible transform are given in the Table 5.6. The method to find i_{left} and i_{right} are discussed in Section 4.5.1 and Figure 4.10.

i_o	i_{left}	i_{right}
even	4	3
odd	3	4

Table 5.6: Extension for reversible transformation using 7-9 filter

5.3.2 Filter Procedure

The filter coefficients defined by the JPEG-2000 for predict and update filters are given by:

α = -1.586 134 342
β = - 0.052 980 118
χ = 0.882 911 075
δ = 0.443 506 852

Since these coefficients are fractional numbers, they were scaled and rounded in order to make the transform operation in integer based. Thus the new integer filter coefficients are:

$$\alpha = \left\lfloor \alpha * scale + \frac{1}{2} \right\rfloor \qquad \beta = \left\lfloor \beta * scale + \frac{1}{2} \right\rfloor$$

$$\chi = \left\lfloor \chi * scale + \frac{1}{2} \right\rfloor \qquad \delta = \left\lfloor \delta * scale + \frac{1}{2} \right\rfloor$$

5.3.3 Transform Algorithm

The following equation describe the sequence of operations performed on the extended signal sets of low-pass signal $\lambda_{0,k}$ and the high-pass signal $\gamma_{0,k}$ (all the notations are the same as from Section 4.1).

Predict I : $$\gamma_{-1,k} = \gamma_{0,k} + \left\lfloor \alpha(\lambda_{0,k} + \lambda_{0,k+1}) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.23)}$$

$$\gamma_{-1,k} = \frac{\gamma_{-1,k}}{scale} \qquad \textit{(Eq 5.24)}$$

Update I: $$\lambda_{-1,k} = \lambda_{0,k} + \left\lfloor \beta(\gamma_{-1,k-1} + \gamma_{-1,k}) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.25)}$$

$$\lambda_{-1,k} = \frac{\lambda_{-1,k}}{scale} \qquad \textit{(Eq 5.26)}$$

Predict II: $$\gamma_{-1,k} = \gamma_{0,k} + \left\lfloor \chi(\lambda_{0,k} + \lambda_{0,k+1}) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.27)}$$

$$\gamma_{-1,k} = \frac{\gamma_{-1,k}}{scale} \qquad \textit{(Eq 5.28)}$$

Update II: $$\lambda_{-1,k} = \lambda_{0,k} + \left\lfloor \delta(\gamma_{-1,k-1} + \gamma_{-1,k}) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.29)}$$

$$\lambda_{-1,k} = \frac{\lambda_{-1,k}}{scale} \qquad \textit{(Eq 5.30)}$$

Scale- transform: $$\gamma_{-1,k} = \varsigma.\gamma_{-1,k} \qquad \textit{(Eq 5.31)}$$

$$\lambda_{-1,k} = {}^{\lambda_{-1,k}}\!/_{\varsigma} \qquad \textit{(Eq 5.32)}$$

The scale-transform value as per given by JPEG-2000 standard is ζ = 1.230 174 105. However, we converted it into approximated integer value before the transform operation, such that;

$$\varsigma = \left\lfloor \varsigma * scale\_transform + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.33)}$$

Hence the final transformed coefficients are determined as:

$$\gamma_{-1,k} = \frac{\gamma_{-1,k}}{scale\_transform} \qquad \textit{(Eq 5.34)}$$

$$\lambda_{-1,k} = \lambda_{-1,k} * scale\_transform \qquad \textit{(Eq 5.35)}$$

These sequences are repeated for 2-D transformation using *square method* described in Section 5.1.3.The number of iterations of 2-D transform for optimum compression and quality is given by the equation;

$$n = \left\lfloor \log_2 (L-1) \right\rfloor \qquad \textit{(Eq 5.36)}$$

The inverse transform follows the reverse sequence of integer operations of forward transform as in every lifting scheme The filter and scaling coefficients are all in the integer format.

Scale-transform:

$$\lambda_{-1,k} = \varsigma.\lambda_{-1,k} \qquad \textit{(Eq 5.37)}$$

$$\gamma_{-1,k} = {}^{\gamma_{-1,k}}\!/_{\varsigma} \qquad \textit{(Eq 5.38)}$$

Update I:

$$\lambda_{-1,k} = \lambda_{0,k} - \left\lfloor \delta(\gamma_{-1,k-1} + \gamma_{-1,k}) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.39)}$$

$$\lambda_{-1,k} = \frac{\lambda_{-1,k}}{scale} \qquad \textit{(Eq 5.40)}$$

Predict I:

$$\gamma_{-1,k} = \gamma_{0,k} - \left\lfloor \chi(\lambda_{0,k} + \lambda_{0,k+1}) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.41)}$$

$$\gamma_{-1,k} = \frac{\gamma_{-1,k}}{scale} \qquad \textit{(Eq 5.42)}$$

Update II:

$$\lambda_{-1,k} = \lambda_{0,k} - \left\lfloor \beta(\gamma_{-1,k-1} + \gamma_{-1,k}) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.43)}$$

$$\lambda_{-1,k} = \frac{\lambda_{-1,k}}{scale} \qquad \textit{(Eq 5.44)}$$

Predict II:

$$\gamma_{-1,k} = \gamma_{0,k} - \left\lfloor \alpha(\lambda_{0,k} + \lambda_{0,k+1}) + \frac{1}{2} \right\rfloor \qquad \textit{(Eq 5.45)}$$

$$\gamma_{-1,k} = \frac{\gamma_{-1,k}}{scale} \qquad \textit{(Eq 5.46)}$$

Finally, the inverse transform coefficients are given by;

$$\lambda_{-1,k} = \frac{\lambda_{-1,k}}{scale\_transform} \qquad \textit{(Eq 5.47)}$$

$$\gamma_{-1,k} = \gamma_{-1,k} * scale\_transform \qquad \textit{(Eq 5.48)}$$

Chapter 6

Lifting Scheme in Reconfigurable Unit

In the algorithmic level, lifting scheme is the fastest implementation of wavelet transform but its purely software implementation is recognized to be a substantial performance bottleneck for the system. In order to accelerate the lifting operation, this Chapter, introduces the concepts of reconfigurable computing and its implementation in lifting scheme.

6.1 Reconfigurable Computing

The main characteristic of *Reconfigurable Computing* (RC) is the presence of hardware that can be reconfigured (reconfigware - RW) to implement specific functionality more suitable for specially tailored hardware than on a general-purpose processor.

6.1.1 Introduction

There are two primary methods in traditional computing for the execution of algorithms. The first method is to use an Application Specific Integrated Circuits (ASICs), which are hardware units designed specifically to perform a given computation. Hence, they are very fast and efficient when executing the exact computation for which they are designed. However, after fabrication the circuit cannot be altered.

The second method is the use of microprocessors, which is a far more flexible solution. By changing the software instructions, the functionality of the system is altered without changing the hardware. However, the downside of this flexibility is that the performance suffers because it is intended to perform reasonably well, for a wide range of applications. Therefore the performance is far below than of an ASIC. The processor must read each instruction from memory, decode it, and then execute. This results in a high execution overhead for each individual operation.

Reconfigurable computing lies in between a hardware only implementation (in ASIC) and software only implementation in terms of flexibility and performance [13]. Like software, reconfigurable hardware is flexible, and can be reconfigured for different applications. Similar to ASIC, the reconfigurable hardware provide a method to map circuits into the hardware. and the high execution speed, operating parallel execution of multiple instructions.

The reconfigurable device itself is a Very Large Scale Integration (VLSI) microchip known as a *Field Programmable Gate Array (FPGA).* These are electrically programmable and can implement complex computations on a single chip, with millions of gates devices currently in production.

6.1.2 Placement of Reconfigurable Unit

There are many designs available for the placement of a reconfigurable computing unit in general-purpose processor architectures. One of the primary variations between these architectures is the degree of coupling (if any) with a host microprocessor.

Programmable logic tends to be inefficient in implementing certain types of operations, such as loop and branch control. In order to most efficiently run an application in a reconfigurable computing system, the areas of the program that cannot be easily mapped to the reconfigurable logic are executed on a host microprocessor. Meanwhile, the areas with a high density of computation that can benefit from implementation in hardware are mapped to the reconfigurable logic [16]. For the systems that use microprocessor in conjunction with the reconfigurable logic, we explain some ways of coupling (see Figure 6.1).

First, reconfigurable hardware can be used solely to provide reconfigurable functional units within a host processor. This allows for a traditional programming environment with the addition of custom instructions that may change over time. Here, the reconfigurable units execute as functional units on the main microprocessor datapath, with registers used to hold the input and output operands.

Second, a reconfigurable unit may be used as a coprocessor. A coprocessor is in general larger than a functional unit, and is able to perform computations without the constant supervision of the host processor. Instead, the processor initializes the reconfigurable hardware and either sends the necessary data to the logic, or provides information on where this data might be found in memory.

Third, an attached reconfigurable processing unit behaves as if it is an additional processor in a multiprocessor system. The host processor's data cache is not visible to the attached reconfigurable processing unit. There is, therefore, a higher delay in communication between the host processor and the reconfigurable hardware, such as when communicating configuration information, input data, and results. However, this type of reconfigurable hardware does allow for a great deal of computation independence, by shifting large chunks of a computation over to the reconfigurable hardware.

Finally, the most loosely coupled form of reconfigurable hardware is that of an external stand-alone processing unit. This type of reconfigurable hardware communicates infrequently with the host processor (if present). This model is similar to that of networked workstations, where processing may occur for very long periods of time without a great deal of communication.

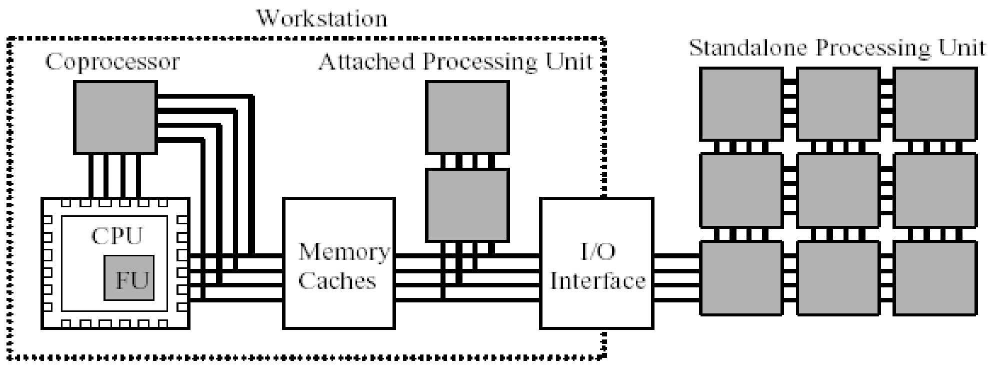

Figure 6.1: Different levels of coupling in a reconfigurable system. The reconfigurable boxes are shaded

Each of these styles has distinct benefits and drawbacks. The tighter the integration of the reconfigurable hardware, the more frequently it can be used within an application or set of applications due to a lower communication overhead. However, the hardware is unable to operate for significant portions of time without intervention from the host processor, and the amount of reconfigurable logic available is often quite limited. The more loosely coupled styles allow for greater parallelism in program execution, but suffer from higher communications overhead. In applications that require a great deal of communication, this can reduce or remove any acceleration benefits gained through this type of reconfigurable hardware.

6.2 Wavelet Transform in Reconfigurable Unit

Wavelet transforms, lifting scheme in particular, discussed in Chapters 3 and 4 are computationally very intensive and when they are used in real time systems, they become even more demanding. The present day general-purpose processors are unable to meet the computational and memory requirements for such applications. Various methods are being applied to enhance its performance such as Single Instruction Multiple Data (SIMD) architectures; multimedia Instruction Set Architecture (ISA) extension like MMX, MDMX; general-purpose processors accelerated with media processors and ASICs performing specialized computations etc.

Since new standards and algorithms are developing rapidly, it is not feasible to change the processors architectures with every new update. In order to speed up the processor performance and make it flexible with different standards, reconfigurable hardware is a suitable option [15][8].

As we discussed before, the software implementation of wavelet transform is computationally very complex and creates a significant bottleneck for practical systems. This can be improved by implementing these algorithms in hardware which can exploit the data reusability and execute transform operations in parallel. In this thesis, the performance of wavelet lifting transforms in software, described in Chapter 5, were compared with reconfigurable FPGA units mapped with the same algorithms[14].

Chapter 7

Methodology

In this Chapter, we will discuss the experimental framework required to carry out the performance analysis of wavelet transform in simulated MIPS processor with and without the reconfigurable hardware unit. We will explain the necessary details and modifications in "*Simple Scalar Toolset Version 3.0* " to implement the hardware unit for wavelet transform, instruction annotation, and benchmarks for analysis.

7.1 SimpleScalar Architectural Design Tool Set

SimpleScalar is a tool set consisting of compiler, assembler, linker, simulation, and visualization tools for MIPS IV architecture [2]. We have extensively used this tool in order to execute the benchmark for the comparison of the performance of software vs. hardware implementation of Wavelet transform. Some of its advantageous features of Simplescalar employed in this thesis are; easy to augment the hardware unit, facility for ISA extension without changing compiler and accuracy on timing information while executing the program.

7.1.1 SimpleScalar

SimpleScalar is a derivative if MIPS-IV architecture that takes binaries compiled on a gcc based compiler for its architecture (see Figure 7.1). It simulates execution in different processors, which are;

- sim-fast for functional simulation
- sim-cheetah and sim-cache for functional cache simulation
- sim-profile for detailed profiles on instruction classes, addresses, symbols etc.
- sim-outorder for timing simulation.

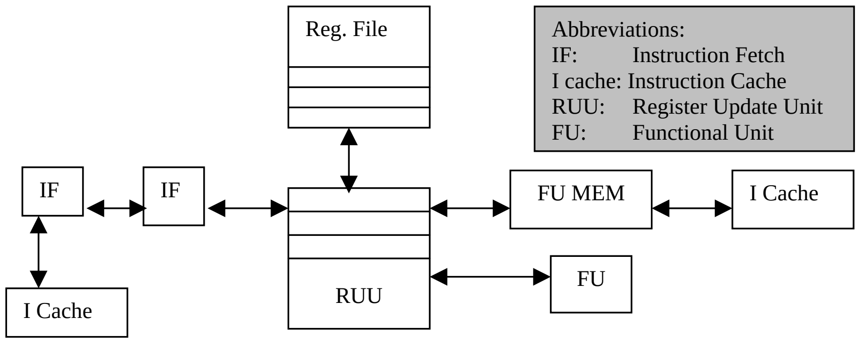

Figure 7.1: Architecture of SimpleScalar

In addition of supporting all MIPS ISA, it has an extended 64-bit instruction encoding including 16 bit annote field (Figure 7.2). The ISA can be modified post compile, with the annotations to instructions in assembly. This allows compile time tool to communicate data to the simulator hardware on demand, which is very useful for synthesizing new instructions without having to change or recompile the assembler.

For example, the annotated instructions can be a written as:

```
la/a  $r2, $r2
```

The annotation in expressed by “/a” which specifies that the first bit of the annotation field is set. It is possible to set bits from 0 to 15 through /a to /p.

In this thesis, the simulator processor “sim-outorder” has been chosen because it is suitable for functional and timing analysis of our unit.

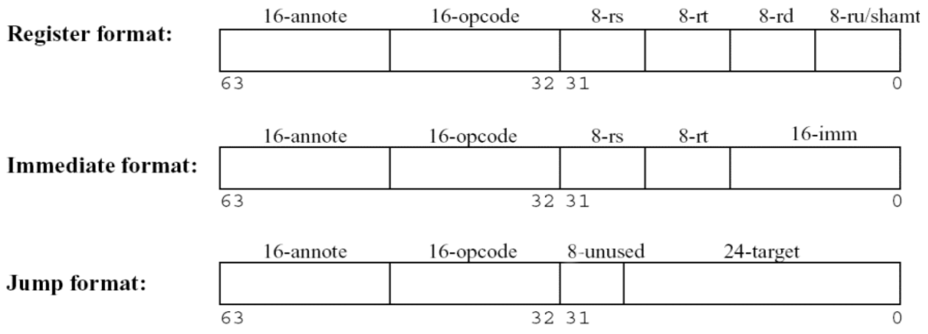

Figure 7.2: SimpleScalar Architecture instruction formats

7.1.2 Sim-Outorder

This processor is an in-order based execution simulator followed by a trace-based out-of-order timing simulator. The instructions are first decoded and executed instantaneously modifying both the Register Update Unit (RUU) and memory. Then they are placed into timing simulator and issued to different execution units. The RUU scheme reorders buffers to automatically rename the registers and to hold the results of pending instructions. Each cycle, the reorder buffer retires the completed instructions in program order to the architected register file.

The processor’s memory employs a load store queue. Store values are put into the queue if the stores are speculative. Loads are dispatched to the memory system when the addresses of all previous stores are known or they could be satisfied by the earlier store value residing in the queue. Speculative loads may generate cache misses but speculative TLB misses stall the pipeline until the branch condition is known.

Sim-outorder simulates a six stages of pipeline (fetch, decode, issue, execute, write-back and commit) as shown in the Figure 7.3. The five stages are implemented with the functions driven by the function sim-main(). The functions and their role in the program are as follows:

ruu_fetch(): this is an execution simulator responsible for fetching instructions from I-cache and place them into a queue for later decoding. Each cycle it fetches only one I-cache line and in case of miss, it blocks the line until it is completed.

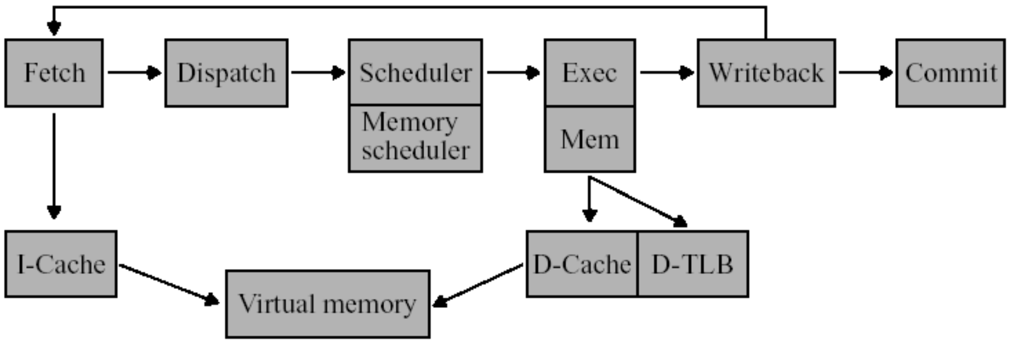

Figure 7.3: Pipeline for Sim-outorder

ruu_dispatch(): this is a stage where new instructions are decoded, executed by the execution based simulator and then placed into RUU structures. According to the width of the target machine, the dispatcher takes instructions from fetch queue in each cycle and places them in the scheduler queue. At this function the instructions make transition from in-order execution based simulator to the out-of -order timing simulator. In this routine the branch mis-predictions are also noted, it is possible because the branch has already been executed.

ruu_issue(): at this stage the ready instructions are issued from the RUU units to the functional units. Each cycle, the instructions are issued out-of-order if they are ready (have all their operands) and if the functional units are not busy. The issue of the ready loads is stalled if there is an earlier store with an unresolved effective address in the load/store queue (LSQ). If the address of the earlier store matches that of the waiting load, the store value is forwarded to the load. Otherwise, the load is sent to the memory system. Finally, it schedules the write-back events using the latency of functional units.

ruu_writeback(): at this stage the instructions are completed in timing simulator and the dependence chain of the instruction output is marked. If the dependent instructions were waiting only for that completion, they are marked as ready to be issued. This function also detects mis-predicted branches as a timing simulator and resets the erroneously issued instruction.

ruu_commit(): this function removes the completed instructions from the RUU and also executes the store instructions, if any, that are at the head of RUU. In other words, it commits the instructions in-order, updates the data caches or memory with store values and handles data TLB miss. When an instruction is committed, no values are saved in RUU or LSQ.

The modifications made in Sim-outorder in order to implement Wavelet Transform hardware unit will be discussed later in Section.7.5.

7.2 MIPS Augmented with Reconfigurable Unit

In order to augment the general-purpose processors with a reconfigurable hardware unit there are several proposed architectures, for example in [8][15]. In most cases, the organization assumes the processor operates like in general-purpose paradigm but it is extended by a reconfigurable unit(s) that speed up the processing. Similar to other functional units, the execution and the reconfiguration are under the control of the “core” processor. Due to the potential re-programmability of the reconfigurable processor, a high flexibility is attained such that tuning the reconfiguration for specific algorithms or for the general-purpose paradigm.

It would have been preferable to compare our approach to existing approaches for reconfigurable processors [8] but due to the absence of tools and precise models we are using a well-known simulation environment, the SimpleScalar Toolset [2].

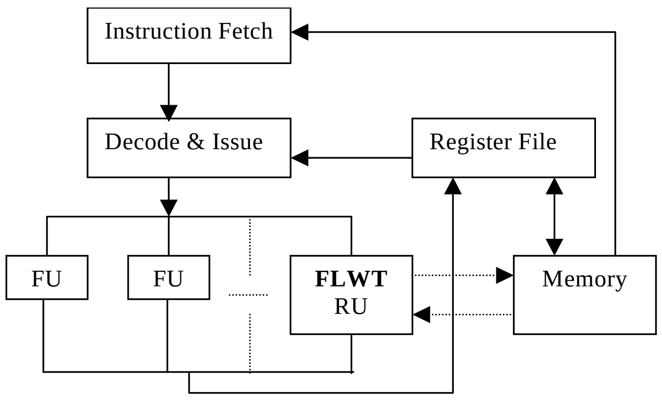

Figure 7.4: Architecture representing MIPS with FLWT unit

A reconfigurable unit, namely "FLWT Unit", was added, as shown in Figure 7.4, to the Sim-outorder processor along with its other existing functional units. The reconfigurable unit was configured only once with the FLWT implementation, hence it can be perceived as a fixed FLWT unit. This unit was simulated as a FPGA unit responsible for executing FLWT operation [17].

7.3 Instructions Annotation for FLWT Operation

In the case of FPGA model [17], the filter and lifting coefficients are placed in the internal RAM, because they are constant values during the FLWT operation, and then the transform operation is conducted by the hardware unit. However, the software model of modified Liftpack, described in Section 5.1, calculates the filter and lifting coefficients and then performs the transform operation. For a realistic simulation of the FPGA model, we need to avoid the calculation of the filter and lifting coefficients by the software and execute the transform operation in FLWT unit. Therefore we introduced 3 new instructions, which will execute a defined set of instructions without affecting the processor execution time. In this thesis, we used the annotation field of the SimpleScalar ISA (Section 7.1) and reclaimed the prevailing instructions, avoiding any alteration in the compiler. These instructions are called *annotation instructions.*

The annotated instructions were introduced into the source code of the Liftpack software, using the assembler directives, such as:

- For calculating filter coefficients
```
__asm__ ("add/a $11,$11,$11")
```
- For calculating lifting coefficients
```
__asm__ ("add/b $11,$11,$11")
```
- For executing the FLWT operation
```
__asm__ ("add/c $11,$11,$11")
```

The annotation was expressed by "/a", "/b" and "/c" with the *add* instruction, which specifies that the first, second and third bit of the annotation field is set, respectively. The rest of the general-purpose registers stated, were chosen arbitrarily, since we were not using any of these registers to pass any data to the hardware.

The necessary data and parameters required for the execution of annotation instruction are transferred from the modified Liftpack software through the general-purpose register using the assembler directive. For example, the value of “data” is passed through the general-purpose register $2:

```
__asm__ ("lw $2, data")
```

The register values passed by the software are received by the Simplescalar and assigned to a variable for further use. For example,

```
Received_data  = GPR(2)
```

While using the annotation instruction, the processor and the FLWT unit share the same register files. This can cause the register values to be overwritten by the FLWT unit. To avoid this situation, we saved all the register values into memory before executing the annotated instruction and afterwards when the execution of the unit is complete the saved register values were restored.

7.4 FLWT Implementation

After close examination of the FLWT algorithm, we programmed the FLWT unit in a way that it can carry out the exact operations and give exact results as being executed in software. The schematic logical block diagram of reconfigurable FLWT unit is given in Figure 7.5.

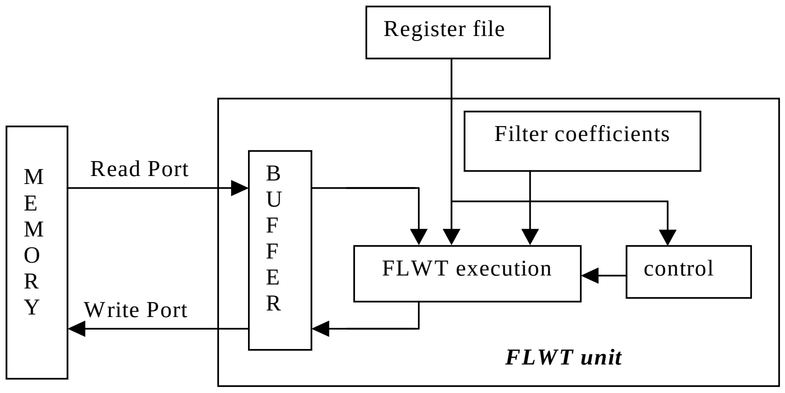

Figure 7.5:Organization of FLWT unit

7.5 Modifications on Sim-outorder Source Code

The new reconfigurable unit loaded with FLWT implementation was added in the resource pool definition of Sim-outorder , which already contained 4 integer ALUs, 1 integer MULT/DIV-unit, 4 FP adders, 1 FP MULT/DIV unit and 2 memory ports(R/W).

In order to execute the FLWT, all the required instructions were added in the function ruu_dispatch (), which also included reading the picture data from memory and writing back the transformed picture data into the memory. The annotation instruction is first detected in the beginning of the function ruu_dispatch() and if it is true the FLWT instructions are decoded and run by the execution based simulator and placed into RUU structures, see Section 7.2.

Since instructions are issued to their corresponding functional units and the Load Store Queue (LSQ) is also resolved at the function ruu_issue [2], we added instructions in this function, to calculate the total latencies on reading or writing the necessary data to/from FLWT unit. Provisions were also made to reserve the FLWT unit while the data were being read or written in order to prevent any unwanted interference with the unit. The memory model of FLWT unit is described in more detail in next Section.

7.6 The Memory Simulation

The SimpleScalar processor simulator has a bus width of 4 bytes and 2 levels of data caches. The cache configuration can be represented in the order of number of blocks, block size and associatively as 128:32:4 and 1024:64:4 for 1st level and 2nd level respectively. While reading the data, if it is a hit, the latency time is of 1 cycle but if it is a miss, then miss penalty is of 6 cycles for 1st level cache and 32 cycles for 2nd level cache. Similarly, the TLB miss will result the latency of 30 cycles.

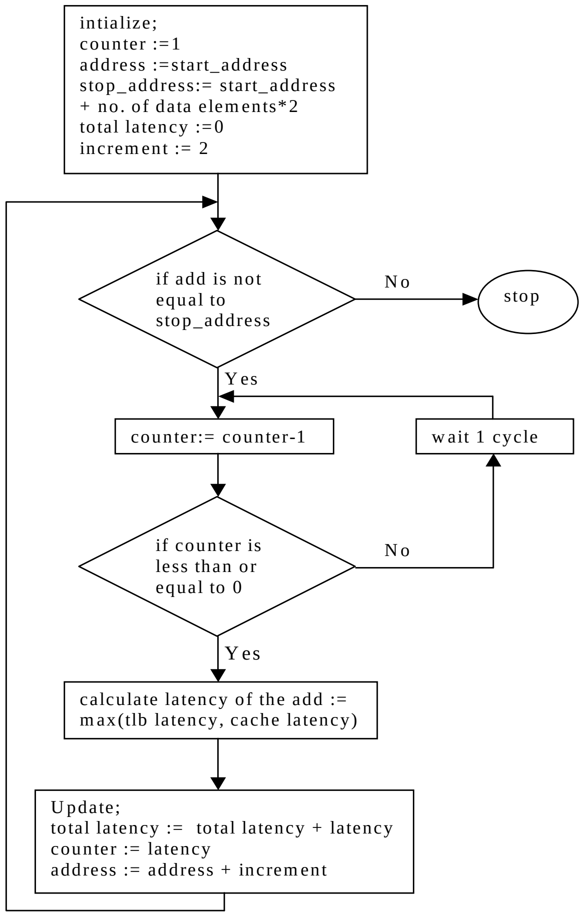
intialize;
counter :=1
address :=start_address
stop_address:= start_address + no. of data elements*2
total latency :=0
increment := 2
if add is not equal to stop_address
No
stop
Yes
counter:= counter-1
wait 1 cycle
if counter is less than or equal to 0
No
Yes
calculate latency of the add := max(tlb latency, cache latency)
Update;
total latency := total latency + latency
counter := latency
address := address + increment

Figure 7.6: Flowchart for calculating the total latency of an image data

The dynamic range of original image data is 0-255, i.e., it can be represented by one byte. But this range would exceed during the transform operation .So we used 2 bytes to represent our picture data. Since the width of the processor data bus is of 4 bytes, 2 data elements were read in FLWT unit through the Read Port of SimpleScalar in 1 cycle. The total load latency was calculated as the summation of individual load latencies, which was taken as the maximum of cache latency and TLB latency. The details of this paradigm are explained by the flowchart in Figure 7.6.

After the transform operation, the data was written back to the host memory through the Write Port of SimpleScalar. Each write latency is of 1 cycle. Since we have the data bus of 32 bits, 2 data elements can be written in each cycle. So the total writing latency can be simply expressed as the cycles numbering half of the data elements in the image.

The Read and Write port of SimpleScalar was used to read and write the data between FLWT unit and the host memory instead of the FLWT unit itself. This has been designed to provide possible parallelism in the device, for example, the transform operation can be initiated after reading some data elements and both R/W operations and transform operations would be simultaneous. However, in our present design, we implemented the 2-D transform where the whole image data is first read into the buffer and after the transform operation it would be written back to the memory. In order to avoid any interference in the unit while reading or writing the data, the FLWT unit is set busy and no operations are allowed.

7.7 Benchmarks

In this thesis we used three benchmarks in order to analyze the performance of MIPS processor with or without the FLWT unit for lifting wavelet transform.

- The first benchmark was the modified Liftpack software based on lifting algorithm described in Chapter 5.1. Since the floating-point operations are very inefficient in hardware implementation, all the operations in the benchmark were made integer based. The transform-scale coefficients and filter coefficients were scaled up and rounded in order to achieve the best integer approximation of floating point operations. However, all the attributes of the software regarding file handling and user interface was not altered.

- The second benchmark was the updated version of the first set (modified Liftpack), in which the newly presented annotated instructions, explained in Sections 7.1 and 7.3, were added. This new instruction executes the benchmark software in the FLWT unit rather than in the MIPS processor, whereas the other attributes of the software like file handling and user interfaces are executed by the processor.

In this thesis, from now onwards, the first benchmark will be addressed as the non-annotated version and second benchmark be addressed as annotated version. The non-annotated version serves as the base performance level, relative to which we measure the performance increase of the annotated version.

- The third benchmark consist of the purely algorithmic implementations, without any kind of user interface and different data file facilities. This include the lifting schemes using different kind of filters namely, polynomial of degree 2-2, Le Gall5-3 and Daubechies 9-7 as described in Chapter 5. These algorithms can be mapped to the FPGA unit and the hardware performance can be determined. So this set of benchmarks were compiled and executed in Simplescalar simulator to analyze the complexity of the schemes in software implementation against the hardware implementation.

As the input for the benchmarks, 3 pictures of different standard dimensions were chosen which were 176*144, 352*288 and 720*560 of the “Lena” image (Figure 3.7).

Chapter 8

Simulation Results Analysis

In this Chapter we will present the simulation framework and analyze the results obtained from simulations of the benchmarks on Sim-outorder. These results will provide the basis for quantitative analysis to determine the performance gain in hardware implementation model (FLWT unit) relative to the software implementation (standard MIPS architecture).

8.1 Simulation Metrics

In order to determine the performance gain provided by the FLWT unit, we compiled both annotated and non-annotated versions of the benchmarks using tools from the SimpleScalar toolset (Version 3.0). Then we executed both versions of benchmarks on the modified Sim-outorder, which supported the new instructions and contained the FLWT unit. The simulation results and analysis on the software and hardware implementation of the FLWT transform will be explained in the following paragraphs.

Our main performance metric is the speedup of FLWT operation, which is given as the ratio between the software execution time and hardware execution time.

$$\text{Speedup} = \frac{\text{software execution time}}{\text{hardware execution time}} \qquad \textit{(Eq 8.1)}$$

8.1.1 Calculation for Software Execution Time

The software execution time is the total amount of time spent to execute the FLWT operation, without considering the time spent by software for user interfaces and file handling procedures. The procedure used to calculate the time for software execution of FLWT is as follows.

1. Determine the total number of execution cycles (TNEC) taken by the non-annotated version of the software in Sim-outorder. This represents the time for transform operations including file handling and user interfaces.
2. Determine TNEC for the annotated version of the software in the modified Sim-outorder, assuming that FLWT unit takes 0 cycles for the execution of transform operation. This TNEC represents the similar operations as in non-annotated version, except that it includes cycles due to the data transfer operations (read/write) by FLWT unit and excludes the transform.
3. The TNEC for actual software implementation is calculated as:

$$\text{TNEC}_{\text{software}} = \text{TNEC}_{\text{non-annotated}} - \text{TNEC}_{\text{annotated}} + \text{R/W cycles} \qquad \textit{(Eq 8.2)}$$

Thus the software execution time is given by:

$$\text{Software execution time (seconds)} = \frac{\text{TNEC}_{\text{software}}}{\text{processor clock cycle}} \qquad \textit{(Eq 8.3)}$$

Where the MIPS processor clock cycle assumed as1 GHz for our simulations.

8.1.2 Calculation for Hardware Execution Time

The hardware execution time is the total time taken by FLWT unit to perform the transform, including the time required to load data into the unit and write it back to the memory.

Hardware execution time = Hardware transform time + Data transfer time

The hardware transform cycles were obtained from a realistic hardware model built in Xilinx FPGA [17].

$$\text{Hardware execution time (seconds)} = \frac{\text{TNEC}_{\text{hardware}}}{\text{hardware unit clock cycle}} \qquad \textit{(Eq 8.4)}$$

Where the hardware clock rate is set as 50MHz for our experiments.

The data transfer time represents the total time necessary to load the picture data into the FPGA and to write back the transformed results into the memory Since the FLWT unit was working on a much lower frequency than the processor, the data transfer rate of processor might exceed the maximum prescribed transfer rate of the FPGA. This well led to have two different data transfer times,

- The data transfer time due to memory organization, see Section 8.5.
- The data transfer time defined by the maximum operating frequency of the hardware (FPGA).

Considering the practical case, the limitation of hardware data transfer time is set by the one, which takes the longer transfer time. In other words,

HW data transfer time = max (data transfer time by processor,
HW data transfer limit) *(Eq 8.5)*

8.2 Simulation Results

The simulation results were obtained upon executing the benchmark software in Sim-outorder. Due to some constraints of the simulator, the results on each execution may deviate in the range of maximum ± 2%.

8.2.1 Performance Analysis of Liftpack with Different Picture Sizes

The simulation results obtained from the Liftpack with different picture sizes are presented in Table 8.1.The parameters and calculated values given in the table are explained as follows:

- **TNEC, hardware (HW cycles):** This is the number of cycles that the hardware module takes to transform the picture and is defined by the architecture, the size of the picture and the filter type and length. *TNEC hardware* does not include cycles needed for data transfer to and from the module. It was taken from a realistic FPGA model [17].

- **Transform time in HW:** This is the time that the hardware module takes to transform the picture, exclusive data transfer time The time to transform in hardware is calculated as:

$$\text{Transform time in HW} = \frac{\text{TNEC}_{\text{Hardware}}}{\text{Hardware clock rate}} \text{ (micro seconds)}$$

- **Picture per second in hardware**: This is the number of pictures per second that can be transformed by the hardware module without considering the data transfer time

$$\text{Pictures per second in HW} = \frac{1}{\text{Transform time in hardware}}$$

- **Cycles per pixel :** This signifies the number of pixels transformed per cycle by the hardware. It is to be noted that number of pixels that are transformed exceed the number of pixels present in an image. For example, as explained in Chapter 5, the minimum level of iterations for a transform operation is given by

$$n = \left\lfloor \log_2 \left(\frac{L-1}{N_{\max}-1} \right) \right\rfloor$$

This means that transform operation are to be repeated *n* times on a image thus increasing the number pixels that undergo transform operation. Theoretically, the number of pixel expansion due to the iterated transforms are calculated to be around 1.5. Therefore the cycles per pixel are given by:

$$\text{Cycles per pixel} = \frac{1}{1.5\text{*length*width}} \text{(cycles)}$$

where length denotes the number of coefficients in a row and width denotes the number of coefficients in column.

- **TNEC, non annotated and annotated :** These are the total processor cycles obtained by executing the non-annotated and annotated versions of the benchmark software discussed in Section 7.7 in Simplescalar simulator. They depend on the corresponding picture size and filter configurations.

- **Data transfer cycles by processor:** This is the total number of cycles spend by processor to transfer (read and write) the data to or from the memory, see Section 7.6. It depends on the cache size, number of pixels to be transferred and the speed of the datapath and processor.

- **Data transfer time by processor:** this can be calculated as:

$$\text{Data transfer time by processor} = \frac{\text{data transfer cycles by processor}}{\text{processor clock rate}} \text{(micro sec)}$$

- **HW data transfer limit:** This limit is due to the maximum allowable data transfer limit imposed in an internal RAM of FPGA due to its operating frequency. The details of these values can be found in [17].

With these results the hardware and software execution time was calculated as explained in Sections 8.1.1 and 8.1.2 respectively.

With the application of FLWT unit, the execution time of wavelet transform becomes faster by 3.4 times for picture size of 176*144, 3.9 times for 352*288 and 5 times for 720*560. The rise in speedup with the picture dimension is because of the efficient use of pipelines by the longer data elements. While setting up the pipelines for 1-D transform, the hardware unit has to wait some cycles irrespective of the input data length. Once the pipeline is filled, it generates one result per cycle. So the longer the input data, the overhead per element will be less, hence faster execution time . All simulations were performed with the polynomial filter degree of 4-4. A graphical representation of the comparison between hardware execution time and software execution time is shown in Figure 8.1.

Picture format	Pixel	176 x 144	352 x 288	720 x 560
TNEC hardware	Cycle	46 000	160 000	586 000
HW clock rate	MHz	50	50	50
Time to transform in HW	µ sec	920	3200	11720
Pictures per second in HW		1087	313	85
Cycles per pixel	Cycle	1.8	1.6	1.5
TNEC, non annotated	Cycle	18 733 563	73 651 640	298 860 436
TNEC,annotated		15 445 553	60 403 941	236 416 348
Data transfer cycles by processor	Cycle	26 915	147 309	856 837
Data transfer time by processor	µ sec	27	147	857
HW data transfer limit	µ sec	42	170	673
TNEC software	Cycle	3 314 925	13 395 008	63 300 925
Processor clock rate in	MHz	1000	1000	1000
Software Execution time	µ sec	3 315	13 395	63 301
Hardware Execution time	µ sec	962	3 370	12 577
Speedup		**3.44**	**3.97**	**5.03**

Table 8.1: Performance analysis on of Liftpack with different picture dimensions

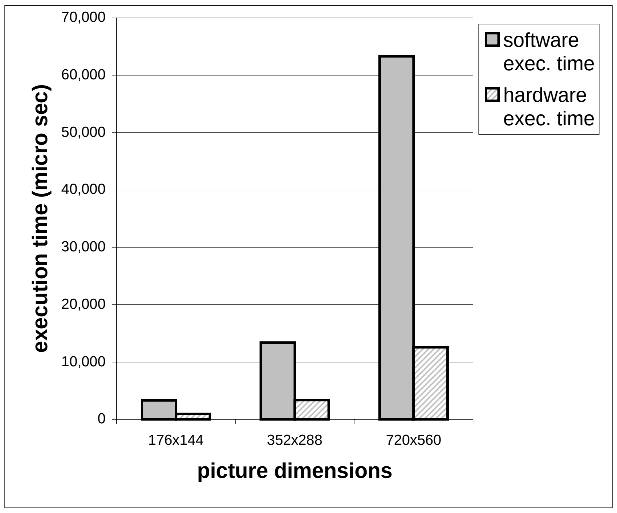

Figure 8.1:Performanc analysis of Liftpack with different picture sizes

8.2.2 Performance Analysis of Liftpack with Different Polynomial Filters

The hardware application becomes more advantageous when the polynomial degrees (number of dual and real vanishing moments) of the filter increase. Table 8.2 presents the simulation results carried out with different polynomial degrees of filter in Liftpack. For the same image of size 352*288. All the analysis were carried out from the recorded simulation results as explained in Section 8.2.1.

The speedup for the filter of 8*8 degree is 6 times while the speedup for the filter of 2*2 degrees is 3 times. It was noted that the hardware execution time does not change significantly in comparison to the software transform time when the degree of polynomial filters increases. In software application when the degree of filter increases, the number of iteration increases and that shoots up the software execution time. However, in hardware once the pipeline for 1-D transform is filled it can generate one result per clock cycle, so increase in the filter polynomials has less effect in the hardware execution time independent of the degree of filter. The comparison between software and hardware execution time is presented in Figure 8.2.

	Unit			
Picture_format	Pixel	352 x 288	352 x 288	352 x 288
Filter of polynomial degrees		2-2	4-4	8-8
TNEC hardware	Cycle	152 000	160 000	175 000
HW clock rate	MHz	50	50	50
Time to transform in HW	µ sec	3040	3 200	3500
Cycles per pixel	Cycle	1.5	1.6	1.7
Pictures per second in HW		329	313	286
TENC software, not annotated	Cycle	69 755 975	73 651640	91 671 177
TENC software, annotated	Cycle	60 071 992	60 403 941	69 757 202
Data transfer cycles by processor	Cycle	147 309	147 309	147 309
Data transfer time by processor	µ sec	147	147	147
HW data transfer limit	µ sec	170	170	170
TENC, software	Cycle	9 831 292	13 395 008	22 061 284
Processor clock rate	MHz	1000	1000	1 000
Software Execution time	µ sec	9 831	13 395	22 061
Hardware Execution time	µ Sec	3 210	3370	3 670
Speedup		**3.06**	**3.97**	**6.01**

Table 8.2: Performance analysis of Liftpack with different polynomial filters

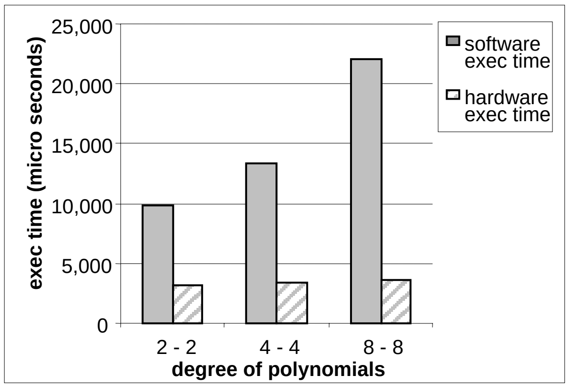

Figure 8.2: Performance comparison of filters with different polynomial degrees

8.2.3 Comparative Analysis of Different Filter Implementations

In Table 8.3, we present the simulation results the wavelet lifting transform using different filters, namely, polynomial filter, Le Gall 3-5 filter and Daubechies 9-7 filters for the same image dimension of 352*288.The description some of the parameters presented in Table 8.3 are given below and the description of the rest is similar as mentioned in Section 8.2.1.

- **TNEC hardware:** The hardware transform time of the polynomial filter type of degree 2-2 was taken from a realistic FPGA model mapped with the same algorithm [17]. This polynomial filter matches closely with the other 2 filter implementations regarding the algorithmic complexity. Since the FPGA model of other these filters was no available, the TNEC hardware execution cycles of the Le Gall 3-5 filter and Daubechies 9-7 filter algorithms, were estimated based on the performance of polynomial 2-2 filter.

- **TNEC software:** The software execution cycles were determined by executing the benchmarks in Simplescalar simulator. The benchmarks were purely algorithmic implementation of Lifting Wavelet transform without any file handling or user interface.

	Unit			
Picture_format	Pixel	352 x 288	352 x 288	352 x 288
Filter type		polynomial: 2-2	LeGall 5-3	Dubechies 9-7
Estimated TNEC hardware	Cycle	152 000	15 2000	152 000
HW clock rate (MHz)	MHz	50	50	50
Time to transform in HW	μ sec	3 040	3 040	3 040
Pictures per second in HW		329	329	329
TNEC software	Cycle	8 833 192	11 860 033	35 990 178
Data transfer cycles by processor		147 309	147 309	147 309
Data transfer time by processor	μ sec	147	147	147
Hardware data transfer limit	μ sec	170	170	170
Processor clock rate	MHz	1000	1000	1000
Software execution time	μ sec	8 833	1 1860	35 990
Hardware execution time	μ sec	3210	3210	3210
Estimated performance ratio of HW vs. SW		**2.75**	**3.69**	**11.21**

Table 8.3:Performance analysis of wavelet transform using different kinds of filter

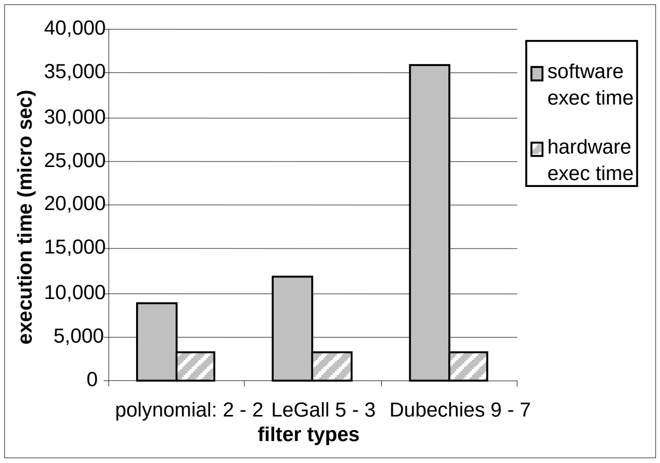

Figure 8.3: Comparison between performance of different filter types

In software implementation, the irreversible wavelet transform was found to be the most computationally intensive. Since the hardware transform time of the reversible and the irreversible transform in FPGA unit was not available, we made a practical assumption on the basis of Liftpack with polynomial filer 2-2 and carried out the analysis. It was found that speedup of Daubechies9-7, Le Gall5-3 and Liftpack with 2-2 polynomial was 11times, 3.6 times and 2.7 times respectively. This is represented in the Figure 8.3

8.3 Effect of Latency on the Performance of Hardware Unit

As discussed in Section 7.5, it costs a significant numbers of cycles while reading the data from or writing into the memory and it negatively affects the performance of the hardware unit. Here we analyze 2 cases using different memory schemes, to get a quantitative measure of the effect of the memory latencies on the hardware execution time.

Case 1: The picture data are read from memory for each execution of 1-D transform and written back after the transform. This requires less internal memory, equal to the length of a line. For instance, the picture size of 352*288 requires only 352 storage elements. However, data have to be read each time for the transform operation.

Case 2: The whole picture data is read at the beginning of the transform and written it back after the completion of 2-D transform. This requires sufficiently large internal memory of the unit to hold the full picture data but less reading/writing with the memory than in case 1.The simulation data for different picture sizes are given in Table 8.4, which can be described as:

- **Total number of pixels to be transferred :** The FLWT algorithm operates multiple times on the same pixel in different levels, according to its multi resolution scheme. So *case 1* requires reading 270270 pixels for a picture format of 352*288 pixels, while *case 2* requires only 101376 pixels for the picture of the same dimension. This data was obtained from the Simplescalar simulator as the number of times the FLWT unit accessed the host memory, during the total transform operation.

- **Total read/write latency (processor cycles):** This is the number of cycles that the external RAM takes to transfer the data to and from the hardware module. These numbers are obtained from the Simplescalar simulator as the number of times the FLWT unit reads from and writes to the memory.

- **Total read/write latency time by processor:** This is the time that the external RAM takes to transfer the data to and from the hardware module.

$$\textbf{Data transfer time} = \frac{\text{Data transfer cycles}}{\text{Processor clock rate}}(\text{micro seconds})$$

- **Hardware data transfer limit:** This limit is set by the maximum speed of the internal FPGA’s RAM. This number was obtained from the FPGA model in [17].

- **Hardware transform Time:** This is the time that the hardware module takes to transform the image, excluding data transfer time. This was obtained from the execution of Liftpack using 4-4 polynomial filter as explained in Section 8.2.2 and Table 8.2.

- **Hardware Execution time:** This is the total time taken by the hardware unit for the wavelet operation including the transform time and data transfer latencies, as described in equation 8.5.

Hardware execution time = Hardware transform time + Data transfer time
= Hardware transform time + max (Total read/write latency time by procesor,Hardware data transfer limit)

	Unit	**Case1**		**Case 2**	
Picture format	Pixel	352*288	720*560	352*288	720*560
Total number of pixels to be transferred	Pixel	270 270	1075 194	101 376	403 200
Total read/write latency cycles	Cycle	1 019 190	10 523 181	147 309	856 873
Processor clock rate	MHz	1000	1000	1000	1000
Total read/write latency time	µ sec	1019	10523	147	856
Hardware data transfer limit	µ sec	450	1790	170	673
Time taken for data transfer	µ sec	**1019**	**10523**	**170**	**856**
Hardware transform time	µ sec	3200	11720	3200	11720
Hardware exe time	µ sec	**4219**	**22243**	**3370**	**12577**

Table 8.4 : Simulation results for case 1 and case 2 using different picture dimensions

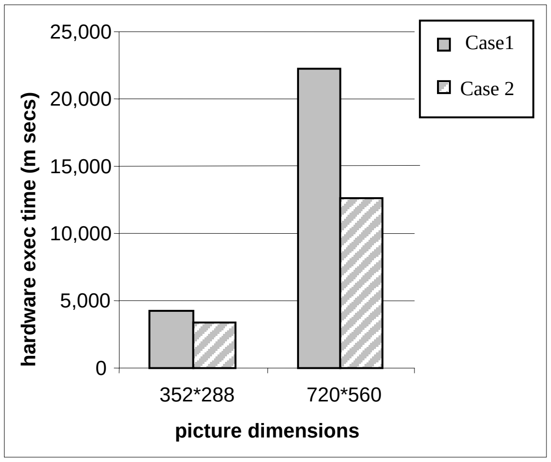

Figure 8.4: Comparison of hardware exec time in case1 and case 2

The results at Table 8.4 shows that the latencies for read/write operations for *case 1* are 6 times more for picture size of 352*288 and 12 times more for picture size of 720*560 in comparison to the *Case 2.*

A major factor for increasing the latency in *case 1*is the inherent cache misses due to accessing the sparse memory locations, which cannot be hold in a cache memory. The data of consecutive rows of an image are generally aligned in a linear array in the external memory. In *case 2*, these data are read sequentially resulting less frequent misses. But, in *case 1* the data are sequentially accessed only during the 1st level horizontal transform, while in the 1st level column transform the distance between two data elements is equal to the length of a row of the picture. With the increase in the level of transform the distance between the data to be read increases more and thus the reading latency.

The effect of reading and writing latencies on the hardware execution time is shown graphically in Figure 8.4.

These results led us to design our FLWT unit to read the whole picture data into the internal memory before beginning the transform operation and write the data back to the memory after the transform is complete.

8.4 Transformation Accuracy

The proposed architecture is based on integer arithmetic for better performance. Although in Lifting scheme integer transform always leads to perfect reconstruction of the original image, rounding errors due to integer operations will cause a certain degree of non-linearity in the transformed signal. We mentioned that scaling factors are used to decrease these non-linearities (increase the accuracy). Simulations pointed out that for polynomial filters of degree less than 4 and 14 bits precision for filter coefficients, an average error of less than 1 is introduced on a dynamic range of 0 to 255 of the input signal. In terms of non-linearity this is equivalent to

$$\textit{Non-linearity} = \frac{\textit{Average error}}{\textit{Dynamic range}} = \frac{1}{256} < 0.4\%$$

Higher orders of the filters cause more non-linearity due to the propagation of error in the calculations.

Chapter 9

Conclusions and Suggestions for Future Research

Wavelet transform allows an efficient de-correlation of data due to its property of space-frequency localization and multi-resolution analysis. Lifting algorithm is the fastest implementation of wavelet transform and it can easily be applied for integer-to-integer transform with perfect reconstruction of the input data. We implemented 3 different lifting schemes for the benchmark software, a modified version of Liftpack [1] and two other standard Fast Lifting Wavelet transform algorithms from JPEG-2000 [14], namely; the reversible wavelet transform (Le Gall 5-3) and the irreversible wavelet transform (Daubechies 9-7).

Despite its many advantageous features, the software implementation of Fast Lifting Wavelet Transform (FLWT) is not applicable for many real time practical systems due to its long execution time. In order to accelerate the FLWT operation, we proposed a hardware unit designed in custom-computing platform. The hardware unit was added as a functional unit in a MIPS based general-purpose processor. A new instruction was introduced as an ISA extension of MIPS architecture for the FLWT hardware implementation. This thesis does not present the hardware implementation but investigates the potential performance of the accelerated hardware implementation compared to the software only approach.

The simulations were performed on a cycle accurate SimpleScalar [2] simulator. The benchmark software were executed in the general MIPS processor and in the FLWT hardware unit, using the new instruction. Base on these simulation results and the hardware execution time from a realistic FPGA model [17], a performance analysis was carried out to evaluate the speed up of the transform due to hardware unit.

We evaluated the speedups for different standard image dimensions, for varying degrees of polynomial filters and for different types of filter implementations. We also analyzed the effect of latencies, encountered by the hardware unit while reading and writing the image data from/to the memory, and its effect on total execution time.

The simulation results show that using the hardware accelerator can substantially improve the performance of the Lifting Wavelet transform. The speedup in the hardware unit was calculated to be 3.4 times for picture size of 176*144, 3.9 times for 352*288 and 5 times for 720*560, on Liftpack with the filter degree 4-4. The speed gain was found to increase with the picture size and the degree of the filter polynomials. This can be attributed to the data reusability and parallelism prevalent in hardware implementation. An estimated analysis of the reversible and irreversible transform using Le Gall 5-3 and Daubechies 9-7 filters showed the performance gain of over 3 times and 11 times respectively against their software versions.

Suggestions for future research

The results attained in this thesis strongly justify the use of hardware units for wavelet transform. Further study and improvements on this topic are highly recommended. Here are suggestions for the future researches:

- The LeGall 5-3 and Daubechies 9-7 are to be designed in FPGA unit for the hardware execution time so that the comparison with their software counterpart would be more precise. The Daubechies 9-7 shows a high speedup potential.

- The reconfiguration time of the FPGA unit has not been taken into consideration in this thesis. Therefore the next step before going into the practical implementation would be to analyze the complexity to reconfigure the unit and its effect on the overall performance.

References

[1] G. Fernandez, S. Periaswamy, and W. Sweldens,"*LIFTPACK: A software package for wavelet transforms using lifting*". Wavelet Applications in Signal and Image Processing IV, Proc. SPIE 2825, 1996, http://www.cse.sc.edu/~fernande/liftpack

[2] Doug Burger, Todd M. Austin, *"The SimpleScalar Tool Set, Version 2.0, Doug Burger"*, http://www.simplescalar.com

[3] Brayn E. Usevitch, *"A tutorial on Modern Lossy Wavelet Image Compression: Foundations of JPEG 2000"*, IEEE Signal Processing Magazine, pp 22-35, September 2001.

[4] Amara Grapes," *An introduction to wavelets*". IEEE Computational Science and Engineering, Summer 1995, vol. 2, num. 2, published by the IEEE Computer Society

[5] Ingrid Daubechies and Wim Sweldens, "*Factoring Wavelet Transforms into Lifting Steps*", *J. Fourier Anal. Appl.*, **4** (no. 3), pp. 247-269, 1998.

[6] Athanassios Skodras, Charilaos Christopoulos, and Touradj Ebrahimi," *The JPEG 2000 Still Image Compression Standard*", IEEE Signal Processing Magazine, pp 36-58, September 2001.

[7] Geoffrey M. Davis, Aria Nosratinia, "*Wavelet-based Image Coding: An Overview", ,*" Applied and Computational Control, Signals, and Circuits, vol. 1, pp. 205-- 269, 1998

[8] J. Hauser and J. Wawrzynek, "*Garp: A MIPS Processor with a Reconfigurable Coprocessor*," Proceedings of the IEEE Symposium of Field-Programmable Custom Computing Machines, pp. 24–33, April 1997.

[9] M. D. Adams and F. Kossentini, "*Reversible integer-to-integer wavelet transforms for image compression: Performance evaluation and analysis*," IEEE Trans. on Image Processing, vol. 9, no. 6, June 2000.

[10] C. Herley and M. Vetterli, "*Orthogonal time-varying filter banks and wavelets*," in Proc. IEEE Int. Symp. Circuits Systems, vol. 1, , May 1993, pp. 391-394.

[11] C. Herley, "*Boundary filters for finite-length signals and time-varying filter banks"*, IEEE Trans. on Circuits and Systems-II: Analog and digital signal processing, vol. 42, no. 2, pp. 102--114, February 1995.

[12] Geert Uytterhoven, "*Wavelets: software and applications*",PhD thesis, April 1999, Catholic University of Leuven.

[13] Vassiliadis, S. Wong, and S. Cotofana. " *The MOLEN rm-coded processor"*. In 11th International Conference on Field Programmable Logic and Applications (FPL), 2001

[14] ISO/IEC FCD 15444-1:2000 V1.0, 16 March 2000), http://www.jpeg.org

[15] Stephan Wong, Sorin Cotofana, and Stamatis Vassiliadis , "*Coarse Reconfigurable Multimedia Unit Extension*", Proceedings of the 9th Euromicro Workshop on Parallel and Distributed Processing PDP 2001.

[16] Parthasarathy Ranganathan, Sarita Adve and Norman P. Jouppiy, "*Performance of Image and Video Processing with General-Purpose Processors and Media ISA Extensions"*, Proceedings of the 26th International Symposium on Computer Architecture. May 1999.

[17] Bahman Zafarifar, "*Micro-codable Discrete Wavelet Transform*", M.Sc. Thesis, Technical University of Delft, 1-68340-28 (2002)-03.

[18] W. Sweldens and P. Schroder, "*Building your own wavelets at home*," Tech. Rep. 1995:5, Industrial Mathematics Initiative, Mathematics Department, University of South Carina, 1995.

Appendix A

1. Description of Source Codes

An electronic version of the source codes, both modified or newly built during the thesis work is submitted in a directory with this report. It is necessary to install the SimpleScalar 3.0 and Liftpack and then replace the files with the modified version presented here for execution. The directories are represented in bold fonts and source code files are represented in italics.

The source code files are included in the following directories:

1. **Simplesim**
2. **Liftpack**
3. **9-7_implementation**
4. **5-3_implementation**
5. **Input_images**

1. Simplesim : The directory Simplesim contains the necessary files required to update SimpleScalar 3.0. This contains 3 source files and a directory;

- *sim-outorder.c* – this is the modified version from the original code obtained SimpleScalar 3.0[2]. A new unit, FLWT, has been added and it executes wavelet transform as a new hardware unit. The theoretical aspects of the modifications are discussed in Section 7.5.
- *machine.c* and *machine.def* – modified from the original files from SimpleScalar 3.0, required to support the FLWT unit added in sim-outorder.

- **files_for_flwt** – this directory contains the files to be added at the project directory with sim-outorder.c, in order to execute the lifting transform (as defined in Liftpack) by the sim-outorder

  mallat.c, core.c, data.c, neville.c, cost.c, details.c, dm.c, error.c, file.c, filter.c, flwd.c, image.c, imgapi.c, lift.c, lu.c, matrix.c, str.c, util.c and their corresponding *header files*

2. Liftpack: here are the modified files from the original Liftpack [1] and a newly made *new_func.c*, which contains the annotated instructions:

- *main.c, pgm.c, txt.c, flwtapi.c, image.c, flwt.h, flwtapi.h and new_func.c*

3. 9-7_implementation: this contains the file *flwt9_7.c,* which is the implementation of Irreversible Wavelet Transform, JPEG-2000 [14]. The algorithm is based on as disussed in Section 5.3.

4. 5-3_implementation: this contains the file *flwt5-3.c,* which is the implementation of Reversible Wavelet Transform, JPEG-2000 [14]. The algorithm is based on as discussed in Section 5.2.

5. Input_Images: here are the input images used for simulation

- *Lena176_144.pgm, Lena352_288.pgm* and *Lena720_560.pgm*

2. Executing Liftpack in sim-outorder

The software Liftpack is first compiled with the gcc compiler of the SimpleScalar Tool Set. In order to select the annotated or non-annotated version of the software, there is an option to define a pre-processor directive EXECUTE ANNOTATION, file *new_func.c*. If it is defined, the FLWT, added in sim-outorder, unit will execute the wavelet transform operation. If it is not defined the program will be simply executed in sim-outorder.

The following instruction is used for executing Liftpack on sim-outorder simulator:

```
>sim-outorder liftpack –t txt –o out.txt Lena176_144.pgm
```

This is the instruction for forward transform where Lena176_144.pgm is the input file and out.txt will be the output file. Similarly, the inverse transform is given by –

```
>sim-outorder liftpack –i –t pgm –o out.pgm out.txt
```

Where out.txt is the input file and the output file would be out.pgm.