Université d'Ottawa · University of Ottawa

Université d'Ottawa • University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

HULUTA, Emanuil

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)

GRADE - DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

Discrete Wavelet Transform Architecture for Image Coding and Decoding

E. Petriu

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

J. Groza

R. Goubran

J.-M. De Koninck, Ph.D.

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

SIGNATURE

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Discrete Wavelet Transform Architecture for Image Coding and Decoding

by

Emanuil Huluta

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the
requirements for the degree of
Master of Applied Sciences
in Electrical Engineering

OTTAWA-CARLETON INSTITUTE FOR ELECTRICAL AND COMPUTER ENGINEERING

School of Information Technology and Engineering
Faculty of Engineering

University of Ottawa, Ontario, CANADA

August 25, 2003

Your file Votre référence
ISBN: 0-612-90082-7
Our file Notre référence
ISBN: 0-612-90082-7

This work is dedicated to the memory of my mother and father,

whose love, guidance and example shaped my engineering career

ABSTRACT

This thesis analyses and implements a *Discrete Wavelet Transform* (DWT) architecture for image processing. The architecture comprises two modules, one for image coding and the other for image decoding. Each module is implemented using a novel *Modified Forward-Backward Register Allocation* (MFBRA) method and accommodates two *Fast Processing Elements* (FPE). The resulting architecture minimizes the hardware required to perform the task together with reduced processing time, rendering the whole structure suitable for real time applications. The whole architecture is implemented and simulated using the Verilog Hardware Description Language.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude and appreciation to my supervisor Dr. Emil M. Petriu for the support, guidance and encouragement he always provided me during the whole period required to complete this work.

My biggest thanks to my wife Camelia whose love and encouragement made the completion of this work possible and also to my beloved daughter Alexandra who patiently waited for me to finalize this, so we can go back to the business of living.

I also thank Zarlink Semiconductor for allowing me the use of their computers and CAD/CAE tools to synthesize and verify the Verilog modules in this work.

LIST OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

2D-DWT	Two-dimension Discrete Wavelet Transform
ASIC	Application Specific Integrated Circuits
CWT	Continuous Wavelet Transform
DC	Direct Current
DCT	Discrete Cosine Transform
DWT	Discrete Wavelet Transform
FBRA	Forward-Backward Register Allocation
FFT	Fast Fourier Transform
FIFO	First In First Out
FSM	Finite State Machine
FPE	Fast Processing Element
HDL	Hardware Description Language
IC	Integrated Circuit
LSB	Least Significant Bits
MAC	Multiply and Accumulate
ME	Multiplier Element
MFBRA	Modified Forward-Backward Register Allocation
MSB	Most Significant Bits
QMF	Quadrature Mirror Filter
RAM	Random Access Memory
RTL	Register Transfer Logic
SNR	Signal to Noise Ratio

Chapter 1 Introduction

Today's growing need to stream video over the Internet, and to reduce the storage space on Multimedia Devices, require higher compression for data representing image and sound. An improved compression reduces implicitly the cost with respect to hardware implementation and bandwidth allocation. Several compression techniques using Discrete Transforms are now implemented in circuits that perform digital image coding. The Discrete Cosine Transform (DCT) is presently the most used technique to accomplish this goal, due to its easy implementation and worldwide interoperability. However, the Discrete Wavelet Transform (DWT) based on time-scale representation provide a better alternative to DCT from many points of view such as adaptive time-frequency windows, lower aliasing distortion, and efficient VLSI implementation.

It is important to mention that Discrete Transforms do not provide bit-rate reduction but rather they produce energy compaction in their coefficient domain. Compression results by cleverly quantifying the high-energy coefficients along with coarse quantifying or dropping of the low energy ones.

1.1 Signal processing using Wavelet Transform

Wavelets decompose the signal at a given level of computation into coarse and detail components at the next level, allowing a recursive structure of computation. This structure can be implemented in a pipelined manner.

The first studies of the Wavelet Transform started in 1976 when Croisier, Esteban, and Galand [7] developed a technique to decompose discrete time signals. At the end of the same year, Crochiere, Weber, and Flanagan [6] did a similar work on coding of speech signals naming their analysis scheme as *Sub-band Coding* [1]. In 1983, Burt defined a technique very similar to *Sub-band Coding* and named it *Pyramidal Coding* [4]-[5], also known as *Multi-resolution Analysis*. Later in 1992, Vetterli and Herley [29] made some improvements to the *Sub-band Coding* scheme, removing the existing redundancy in the *Pyramidal Coding* scheme. Recent studies of Mallat [20] and Daubechies [11]-[12] provided higher compactness to DWT implementation, rendering it as suitable replacement for DCT.

The DWT is derived from the Continuous Wavelet Transform (CWT). CWT represents a correlation between the signal and the wavelet function at different scales and translations. The scale is the inverse of frequency and is used as a measure of similarity. The wavelets are obtained by changing the scale of the analysis window, and by translating the window in time according to the expression:

$$\gamma(s,\tau) = \int f(t)\, \Psi^*_{s,\tau}(t)\, dt \quad \textbf{(1.1)}$$

The CWT coefficients $\gamma(s,\tau)$ are calculated by multiplying the complex conjugated wavelet function $\Psi^*_{s,\tau}(t)$ with the signal $f(t)$, and integrating over all times.

CWT is sampled with respect to its translation and scale variables, resulting the *discrete wavelet* that substitutes s with ks_0 and τ with $k\tau_0$ in the given expression. A *dyadic grid* is

used to sample the coefficients with respect to both variables. The dyadic grid is produced by adopting a scale factor $s_0^k = 2^k$ and a translation factor $\tau_0 = 1$. This means that the signal spectrum is successively divided in two obtaining **1/2, 1/4, 1/8, ... $1/2^n$**, ... portions of the original one. The full bandwidth is covered only if $n \rightarrow \infty$.

A low-pass filter called *scaling function* is introduced to replace the wavelets from a given order k to ∞. In this situation, the discrete wavelets can be implemented in a sequence of stages called *filter-bank*. Each stage splits its input spectrum in two equal parts, one resulting from a *wavelet function* and the other from a *scaling function*

The signal spectrum is limited in frequency (between zero and a positive value) since the signal observed is limited in time. The discrete wavelet has a finite number of coefficients but is still a continuous function with respect to time.

The Discrete Wavelet Transform is obtained by sampling in time and digitally quantifying the *discrete wavelet* coefficients. The filtering operation in the case of DWT is performed using digital filters. The analysis of the signal at each stage is accomplished by using a low-pass filter **H** as *scaling function* and a high-pass filter **G** as *wavelet function*. The low-pass filter feeds the signal for the next stage. This structure is named *sub-band coding* and is presented below.

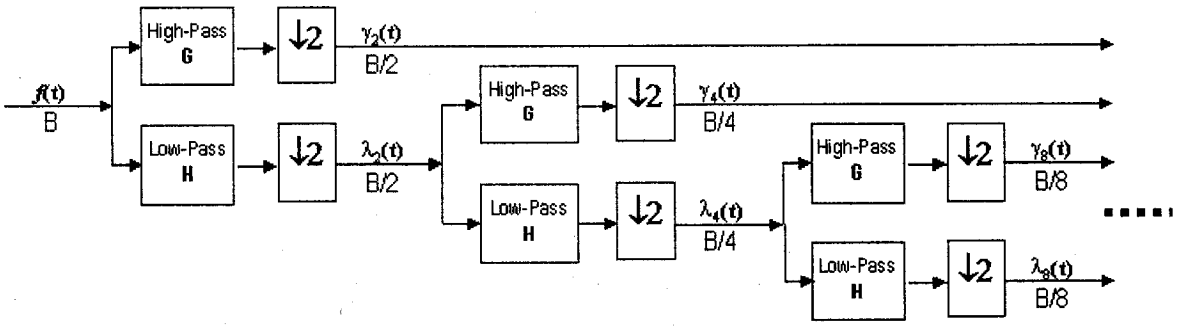

Figure 1. DWT recursive structure

Each stage of the DWT recursive structure divides the original bandwidth $\mathbf{B/2^K}$ in two equal sub-bands of width $\mathbf{B/2^{K+1}}$. The resulting sub-bands are represented by two of the sets of coefficients: $\lambda_j(t)$ from the scaling function and $\gamma_j(t)$ from the wavelet function. Each filter is followed by a down sampling stage that discards each second sample of the filtered signal. This down sampling is possible since dividing the signal bandwidth in two equal parts is equivalent to reducing its minimal Nyquist sampling rate to half for each sub-band.

If $f(t)$ has $\mathbf{2^N}$ samples, this structure can have **N-1** stages, each stage down-sampling the results by 2. The last stage will generate only two coefficients out the total of $\mathbf{2^N}$. In many practical cases, the number of stages is smaller than the maximum possible value of **N-1**. It is important to note that the total number of coefficients resulting from DWT analysis is always equal to the number of the input samples, and is independent of the number of stages used.

The reconstruction or synthesis of the original signal is possible by mirroring the analysis structure if the filters **G** and **H** are built from *orthonormal* bases. These pairs of filters are known as Quadrature Mirror Filters (QMF).

1.2 Thesis Motivation

Video image processing requires analyzing the signal simultaneously in time and frequency. The lack of time resolution characteristic to Fourier Transform has generated the development of other solutions. The Discrete Wavelet Transform is one of the best to overcome this problem due to its relatively easy and straightforward hardware implementation.

Several architectures of DWT such as "DWT systolic array" in [15], "folded wavelet" and "digit serial wavelet" from [24] are known from the literature. Each mentioned architecture has its own advantages and disadvantages but the common limitation for all is the lack of capability to process high-resolution video signal in real time.

The motivation for this thesis was to provide the architecture capable to overcome the above-mentioned limitation by implementing a fast parallel computational unit inside an improved architecture. The improvement in architecture was achieved by developing a new design method.

The DWT architecture is implemented using a 0.35μm Application Specific Integrated Circuit (ASIC) CMOS technology in order to achieve high computational speed required by real time video processing.

1.3 Thesis Organization and Contributions

Chapter 2 provides an overview of the Wavelet Transform. It starts with the Continuous Wavelet Transform describing the necessary steps to required to create the Discrete Wavelet Transform that can be easy implemented in hardware. This chapter includes important notes regarding CWT properties, filter banks, scaling function, sub-band coding, and Quadrature Mirror Filters. The signal reconstruction using DWT is also described.

Chapter 2 concludes highlighting the elements necessary to an optimal implementation of a complete DWT architecture including both analysis and synthesis.

Chapter 3 describes the implementation of both DWT Analysis and DWT Synthesis. This chapter also provides a detailed presentation of the novel Modified Forward Backward Register Allocation method. An original Fast Processing Element (FPE) that avoids arithmetic overflow is also described. The two arithmetic elements of FPE, the radix 2 multiplier and the carry select adder are briefly discussed at the end of Chapter 3.

Chapter 4 presents the characteristics of the synthesized structure resulted from the Verilog HDL code. The simulations resulting from the synthesized code are further discussed, and useful conclusions related to the implementation are drawn.

The contributions of this thesis are:

1. Description of a new method called *Modified Forward-Backward Register Allocation* that allows the designer to eliminate more than 20 % of the hardware usually required

for this type of architecture (see paragraph 3.2). This reduction is achieved by adding new elements to the tabular description of the DWT *Folded Architecture*. MFBRA is explained by implementing a DWT Analysis module for image coding, and a DWT Synthesis module for image decoding. This new method also offers a more comprehensive way of designing architectures that use recursive algorithms.

2. Design of a specialized architecture that provides a real time processor for high-resolution video signal (see Chapter 3). This processor is intended to improve the image quality by replacing the Discrete Cosine Transform structures presently used in most of the image processing circuits.

3. Design of an original Fast Processing Element that reduces the signal noise due to fixed-point computation and avoids arithmetic overflow (see paragraph 3.4).

4. Description of a novel method that *normalizes* at analysis and *de-normalizes* at synthesis, the Daubechies filter coefficients [8], resulting an improved precision of DWT computation without increasing the size of the FPE accumulator (see paragraph 3.6).

Chapter 2 Discrete Wavelet Transform

The wavelet transform uses a fully scalable window that is shifted along the signal, resulting a collection of spectrums evaluated for each position in time. This collection of representations is also called *Multi-resolution Analysis*. In the case of Wavelet Transform the term *time-frequency* representation is replaced by *time-scale* representation, where *scale* symbolizes *1/frequency*.

The *Multi-resolution Analysis* provides a useful description of a signal by including simultaneously coarse and detail information about its structure. This makes wavelet analysis suitable for many applications such as image processing and pattern recognition.

2.1 Continuous Wavelet Transform

The wavelet analysis is performed using a function called *Continuous Wavelet Transform* (CWT) resulting an infinite number of coefficients $\gamma(s,\tau)$ expressed as:

$$\gamma(s,\tau) = \int f(t)\psi^*_{s,\tau}(t)dt \quad \textbf{(2.1)}$$

where "*" denotes complex conjugation. This expression describes how a function ***f(t)*** is decomposed using a set of basis functions $\Psi_{s,\tau}(t)$ called wavelets. The wavelets are generated from a single basic function called the *mother wavelet,* through scaling and translation according to the expression:

$$\psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-\tau}{s}\right) \quad \textbf{(2.2)}$$

where s is the *scale* factor and τ is the *translation* factor. The term $1/s^{1/2}$ provides the *energy normalization* corresponding to different scales. The inverse wavelet transform is a double integral with respect to both variables s and τ according to the expression:

$$f(t) = \int\int \gamma(s,\tau)\psi_{s,\tau}(t)d\tau ds \quad (2.3)$$

2.2 CWT Properties

Wavelets have many properties and the most two important ones gave wavelets their name. The first one is the *admissibility condition* that guarantees the analysis and synthesis of a signal without loss of information:

$$\int \frac{|\Psi(\omega)|^2}{|\omega|} d\omega < +\infty \quad (2.4)$$

where $\Psi(\omega)$ represents the Fourier Transform of $\psi(t)$, and has to satisfy the condition:

$$|\Psi(\omega)|^2\Big|_{\omega=0} = 0 \quad (2.5)$$

The null value for at null frequency implies that wavelets have band-pass spectrum, and translates into null average of $\psi(t)$ in the time domain. This means that $\psi(t)$ must be a ***wave*** (symmetrical to origin).

The second important ones are called the *regularity conditions.* The equation (2.1) shows that the CWT of one-dimensional function is two-dimensional. This means that the time-bandwidth product of the CWT is the square root of the input signal that translates into the square root of memory space necessary to store the transform. This impediment is solved by forcing CWT to quickly decrease proportionally with scale s. The *regularity conditions*

provide smoothness and concentration in both time and frequency domain of the wavelet function. The concept of vanishing moments is used to explain the regularity.

The expansion of CWT into Taylor series of order **n** at **t = 0** gives:

$$\gamma(s,0) = \frac{1}{\sqrt{s}}\left[\sum_{p=0}^{n} f^{(p)}(0)\int \frac{t^p}{p!}\psi\left(\frac{t}{s}\right)dt + O(n+1)\right] \quad \textbf{(2.6)}$$

where $f^{(p)}$ represents the p^{th} derivative of f and $O(n+1)$ represents the remainder of the expansion. Defining the *moment* of wavelet M_p as:

$$M_p = \int t^p \psi(t)dt \quad \textbf{(2.7)}$$

the Taylor series expansion from **(2.6)** is rewritten as:

$$\gamma(s,0) = \frac{1}{\sqrt{s}}\left[f(0)M_0 s + \frac{f^{(1)}(0)}{1!}M_1 s^2 + \frac{f^{(2)}(0)}{2!}M_2 s^3 + \ldots + \frac{f^{(n)}(0)}{n!}M_n s^{n+1} + O(s^{n+2})\right] \quad \textbf{(2.8)}$$

The admissibility condition makes the moment $M_0 = 0$. If the other moments up to M_n are also made zero, the wavelet transform coefficients $\gamma(s,\tau)$ will decay or vanish with the power of s^{n+2} for any smooth signal $f(t)$. For this reason, the *regularity conditions* are also known as *vanishing moments*. A wavelet of order **N** is a wavelet that has **N** vanishing moments.

The fast decay of CWT coefficients gives the term "***let***" that completes the name ***wave-let*** for this type of transform.

2.3 Discrete Wavelets

The Continuous Wavelet Transform is not suitable to be used as a signal-processing tool due to three major disadvantages. The first one is CWT infinite number of scaling and translations that produces an infinite number of coefficients. The second one is the redundancy of these coefficients. The third one is the lack of analytical solution of CWT that makes this transform unsuitable for computer algorithms.

The *discrete wavelets* have been introduced to reduce the number of coefficients by scaling and translating the wavelets in discrete steps according to the formula:

$$\Psi_{j,k}(t) = \frac{1}{\sqrt{s_0^j}} \Psi\left(\frac{t - k\tau_0 s_0^j}{s_0^j} \right) \quad \textbf{(2.9)}$$

The discrete wavelet is still a piecewise continuous function having j and k integers, a fixed dilatation step $s_0 > 1$, and a translation factor τ_0 depending on s_0. A useful value for s_0 is 2 yielding a *dyadic sampling* (2^N) on the frequency axis. This power of 2 sampling matches the octave scale of music sounds, and is easy to use in the computers. The translation factor $\tau_0 = 1$ provide *dyadic sampling* on the time axis. Figure 2 below shows the placement of discrete wavelets on a dyadic grid.

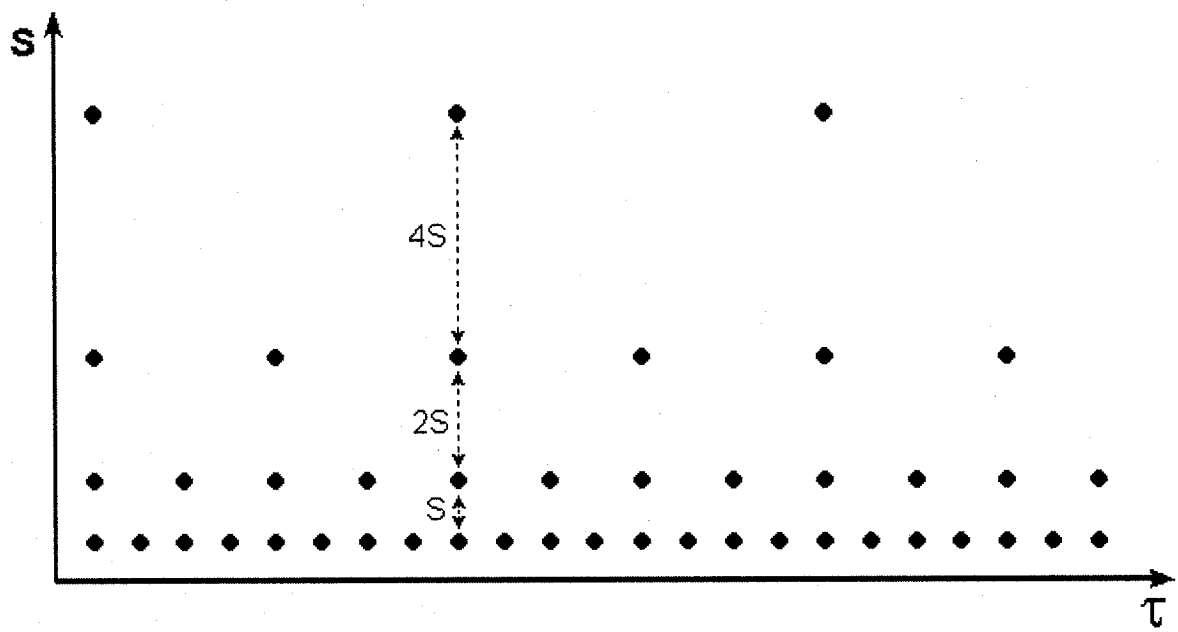

Figure 2. Dyadic grid for discrete wavelet coefficients

The discrete wavelet transform of a continuous signal is a series of wavelet coefficients named *wavelet series decomposition*. The necessary and sufficient condition to reconstruct a signal from its wavelet series decomposition is given by I. Daubechies in [11]. This condition states that the energy of the wavelet coefficients has to be limited by two positive bounds:

$$A\|f\|^2 \leq \sum_{j,k} \left|\langle f, \psi_{j,k} \rangle\right|^2 \leq B\|f\|^2 \quad \textbf{(2.10)}$$

where $\|f\|^2$ is the energy of $f(t)$, $A > 0$, $B < +\infty$ and both A and B are independent of $f(t)$. When this condition is satisfied, the family of basis wavelets $\psi_{jk}(t)$ is called *frame* having A and B as limits. When A = B the frame is called *tight* and the corresponding discrete wavelets constitute an *orthonormal* basis [8], [10]. When $A \neq B$ the *frame* becomes *dual* and the decomposition wavelet is different from the reconstruction wavelet.

The discrete wavelets are orthogonal to their own dilatations and translations if their mother wavelet satisfies the condition:

$$\int \psi_{j,k}(t)\psi^{*}_{m,n}(t)dt = \begin{cases} 1 & if \quad j=m \quad and \quad k=n \\ 0 & otherwise \end{cases} \qquad \textbf{(2.11)}$$

This condition is satisfied when the signal has a finite energy (expression 2.12), or in other terms, the L^2norm of ***f(t)*** is finite:

$$\int_0^{\infty} |f(t)|^2 dt < \infty \qquad \textbf{(2.12)}$$

2.4 Wavelet Filter Bank

When condition (2.11) is satisfied, an arbitrary signal can be reconstructed by summing its orthogonal wavelet basis functions weighted by the wavelet coefficients: The equation (2.13) represents the *Inverse Discrete Wavelet Transform.* Equation (2.13) is the discrete version of the double integral Inverse CWT given in (2.3).

$$f(t) = \sum_{j,k} \gamma(j,k)\psi_{j,k}(t) \qquad \textbf{(2.13)}$$

The condition (2.5) stated above suggests that wavelet has a band-pass type of spectrum. Fourier analysis demonstrates that a compression of signal in time with a factor **K** is equivalent to stretching the spectrum and shifting it upwards with the same factor **K**. Applying this property to wavelets means that a time compression by a factor of 2 will stretch the frequency spectrum by 2 and shift all the frequency components also by 2. Using this property, the finite spectrum of a given signal can be covered with a sum of dilated wavelet spectrums corresponding to the time domain translated wavelets. Some

overlap of adjacent wavelet spectrums is required (see Figure 3) to obtain a good coverage of the original signal spectrum. This is achieved by carefully designing the wavelets.

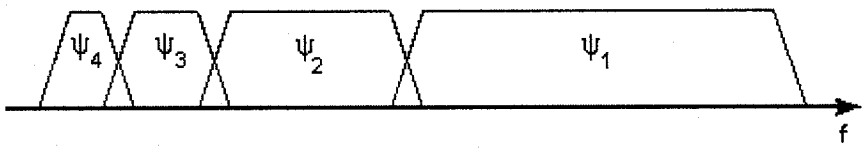

Figure 3. Wavelet filter bank

The resulting series of filters described in is called *filter bank* and has the important property that the ratio between the center frequency and the width of spectrum is the same for all the wavelets. This ratio represents the quality factor Q of each filter, resulting a *constant Q filter bank.*

2.5 Scaling Function and Wavelets

The filter bank described above will still have an infinite number of components obtained by continuously dividing down the spectrum to an infinite small half. This problem is solved by replacing the infinite number of filters existing from a certain $j = n+2$ order band-pass to zero frequency by a low pass filter described by a *scaling function.* The *scaling function* has been introduced by Mallat in [19].

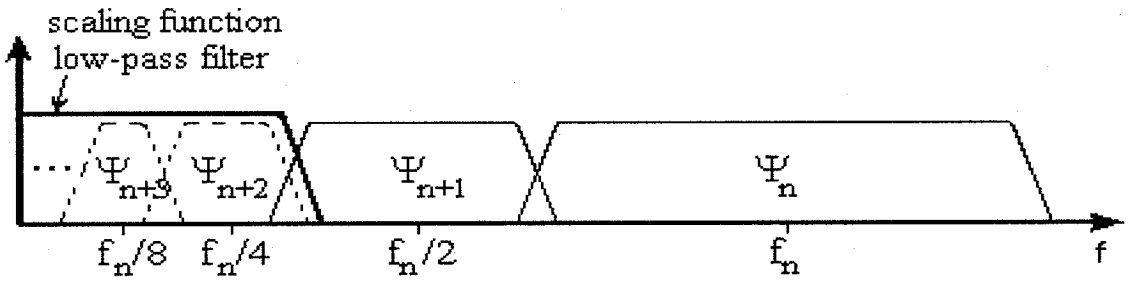

Figure 4. Scaling function for a filter bank

The scaling function resulting from replacing the infinite set of low frequency wavelets can be written in the same manner as the Inverse Discrete Wavelet Transform [17]:

$$\varphi(t) = \sum_{j,k} \gamma(j,k)\psi_{j,k}(t) \quad \textbf{(2.14)}$$

This function sets the lower bound of the wavelet representation while the upper bound is given by the finite duration of the signal analyzed. The width of the scaling function spectrum represents an important parameter that has to be carefully designed to balance the remaining wavelet coefficients. This problem is automatically solved by the Discrete Wavelet Transform.

The spectrum of the scaling function satisfies the condition (2.15) similar to CWT *admissibility condition* (2.5):

$$\int \varphi(t)dt = 1 \quad \textbf{(2.15)}$$

2.6 Sub-band Coding

The discrete wavelet built as a finite filter bank is now suitable to be implemented in a computational structure. Two possible implementations are discussed, one having a *set of band-pass filters* and another using a technique called *sub-band coding*.

The *set of band pass filters* has the advantage of flexibility in analyzing the interesting portions of signal spectrum. The disadvantage of this architecture is that each filter has a dedicated design making the whole structure difficult to implement.

The *sub-band coding* is a multi stage filtering structure. At each stage, the signal spectrum is split in two equal portions; a high-pass and a low pass one. Always the low-pass portion contains more information than the high pass one. For this reason, the following stage splits in two only the low-pass portion, leaving the high-pass unchanged. An *iterated filter bank* is obtained by repeating this algorithm (see Figure 5). The number of splitting stages is limited by the available computation power. The advantage of this method is that it can be implemented in a recursive manner by designing only two filters, a high-pass and a low-pass one. The disadvantage is that the signal spectrum has a fixed dyadic partition. The high-pass filter is in fact a band-pass filter since the upper part of the spectrum is inherently limited by the nature of the wavelet.

According to the structure described in Figure 4, the wavelets are the band-pass portions of the higher frequencies, and the scaling function is the low-pass portion. Each band-pass portion has double bandwidth with respect to its lower frequency neighbor.

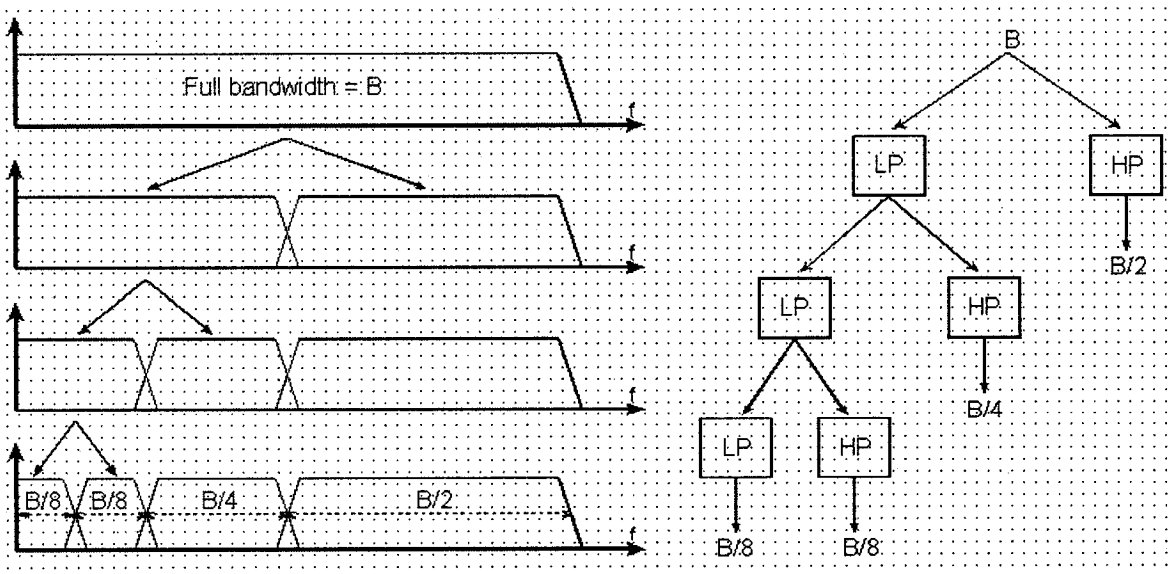

Figure 5. Iterated Filter Bank Structure

In this way, the wavelet transform is implemented using a sub-band coding scheme

2.7 DWT Analysis

The structure described in Figure 5 represents the *DWT Analysis* and is now easy to implement in a digital signal processing architecture using numerical low pass and high-pass filters. Any numerical filter can be implemented using an Infinite Impulse Response (IIR) structure or a Finite Impulse Response (FIR) structure.

The target is to obtain a fast computational architecture that can be achieved by sampling in time and numerically quantifying the input signal. An efficient numerical representation for high speed processing is fixed point samples. The IIR filter is not suitable for fixed-point sample representation due to its feedback structure that may lead to arithmetic overflow or underflow, altering drastically the quality of the filtered signal. FIR filters are considered instead by the majority of the wavelet applications known in the literature, and a strong developed mathematical apparatus is supporting their efficient implementation.

It is known from the signal theory that filtering a signal represents the convolution in time between the signal and the filter function. In the case of discrete representation, the FIR filtering is given by the convolution:

$$\mathbf{f[n]^{*}\cdot h[n] = h[n]^{*}\cdot f[n] = \sum_{k} h[n]\cdot f[n-k],} \qquad \mathbf{k=0..L-1} \qquad \mathbf{(2.16)}$$

where L is the length representing the number of taps of the FIR filter

A discrete signal sampled with a normalized pulsation $\omega_n = 2\pi$ has the bandwidth limited between 0 and π according to Nyquist rule. In this case, the DWT will start the dyadic bandwidth division with $B = \pi$. The first stage of DWT produces two equal sub-bands, one from 0 to $\pi/2$ and the other from $\pi/2$ to π. Applying Nyquist rule again, the number of coefficients required to fully characterize each sub-band is now half of the original number of samples. Thus, every second coefficient resulting from each filter is redundant and can be discarded without affecting the quality of the transform. This operation is named sub-sampling and the resulting DWT coefficients are the result of the convolution. Dropping every second sample is equivalent with replacing the index **n** with **2n** in (2.16), resulting the new relation:

$$\lambda[2n] = \sum_k h[k] \cdot f[2n-k], \quad k=0..L-1 \qquad (2.17)$$

The relation given in (2.17) can now be generalized to characterize any stage of the DWT. In the following relations, the particular case of input samples ***f*[n]** has been replaced with the general case of *scaling coefficients* resulting from the low pass filter of the previous stage **n-1**. The DWT coefficients have been grouped *in scaling coefficients* (2.18) resulting from the low pass filter, and *wavelet coefficients* (19) resulting from the low pass filter:

$$\lambda[2n] = \sum_k h[k] \cdot \lambda[2n-k], \quad k=0..L-1 \qquad (2.18)$$

$$\gamma[2n] = \sum_k g[k] \cdot \lambda[2n-k], \quad k=0..L-1 \qquad (2.19)$$

The structure described by the equations (2.18) and (2.19) is now easy to implement, and is shown in Figure 6 below.

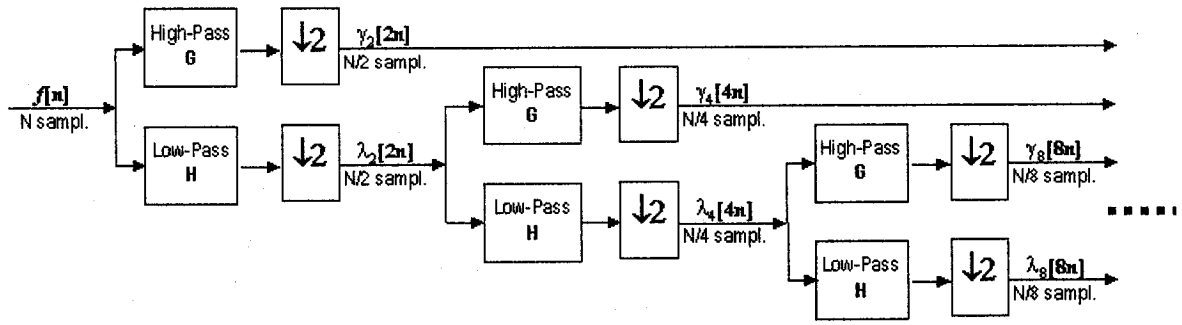

Figure 6. DWT Analysis structure

An example is given assuming that the original signal ***f*[n]** has $\mathbf{N} = 2^8 = 256$ sample points and covers a frequency band of **0** to π rad/s. The first decomposition level passes the signal through its high-pass and low-pass filters, and then sub-samples it by two. The output of the high-pass filter has **N/2** = 128 points representing the wavelet coefficients **γ[2n],** and covers the frequencies from **π/2** to π rad/s. The output of the low-pass filter has also 128 points representing the scaling function coefficients **λ[2n]**, and it covers the other half of the frequency band from **0** to **π/2** rad/s. The scaling function coefficients **λ[2*n*]** are then passed through the same low-pass and high-pass filters for further decomposition. The output of the second low-pass filter followed by sub-sampling has 64 coefficients **γ[4n]** covering the frequency band from **0** to **π/4** rad/s, and the output of the second high-pass filter followed by sub-sampling has 64 coefficients **λ[4n]** covering the frequency band of **π/4** to **π/2** rad/s. In the given example, the process continues for 8 - 1 = 7 stages, until two coefficients (one for each filter) are left. The total number representing the DWT coefficients is 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 + 1 = 256 which is the same as the original signal.

It is important to mention that the number **N** of samples processed by DWT has to be a power of 2, or at least to satisfy the relation:

$$\mathbf{N = 2^L \cdot M, \quad M \geq L} \qquad \mathbf{(2.20)}$$

where L is the number of stages and M is an integer number

2.8 Quadrature Mirror Filters

The high-pass filter **G** and the low pass filter **H** are built from orthonormal bases and they belong as conjugated pair to a special category of filters called Conjugated or Quadrature Mirror Filters (QMF). The QMF are commonly used to divide the full bandwidth **[0–π]** of a sampled signal in two equal sub-bands **[0–π/2]** and **[π/2–π]**. The FIR high-pass coefficients **g[k]** and low-pass coefficients **h[k]** of such filters satisfy the relation:

$$\mathbf{g[L-k] = (-1)^k \cdot h[k] \quad K=0..L-1 \quad (2.21)}$$

This relation simplifies the design of DWT architecture to one set of FIR filter coefficients, because the second set is obtained very easy from the first one using (2.21).

The necessary condition for perfect reconstruction of the DWT coded signal, is the use of Quadrature Mirror Filters.

2.9 DWT Synthesis

The reconstruction or synthesis of the original signal is done by mirroring the analysis structure with respect to all its components. The DWT synthesis mirrored from DWT

analysis, up-samples by two the analysis coefficients, then processes the up-sampled coefficients with mirrored filters, and sums the result of each pair of filters. Up sampling is done by adding a null coefficient after each DWT analysis coefficient. The mirrored high-pass filter and low-pass filter have respective the same coefficients as DWT analysis filters, but the order is reversed:

$$\mathbf{g'[k] = g\,[L-k],} \quad \mathbf{h'[k] = h[L-k]} \qquad \mathbf{(2.22)}$$

where **L** is the number of taps. The reconstruction equation is:

$$\mathbf{\lambda'[n] = \sum_k \{\, h[L-k] \cdot \lambda_U[n-k] + g[L-k] \cdot \gamma_U[n-k] \,\},\ \ k=0..L-1} \qquad \mathbf{(2.23)}$$

where λ_U**[n-k]** and γ_U**[n-k]** are the corresponding up-sampled versions of λ**[2n-k]** and γ**[2n-k]**.

The DWT synthesis structure resulting from (2.23) is shown in Figure 7,where each pair of filters together with the summing function form a single stage.

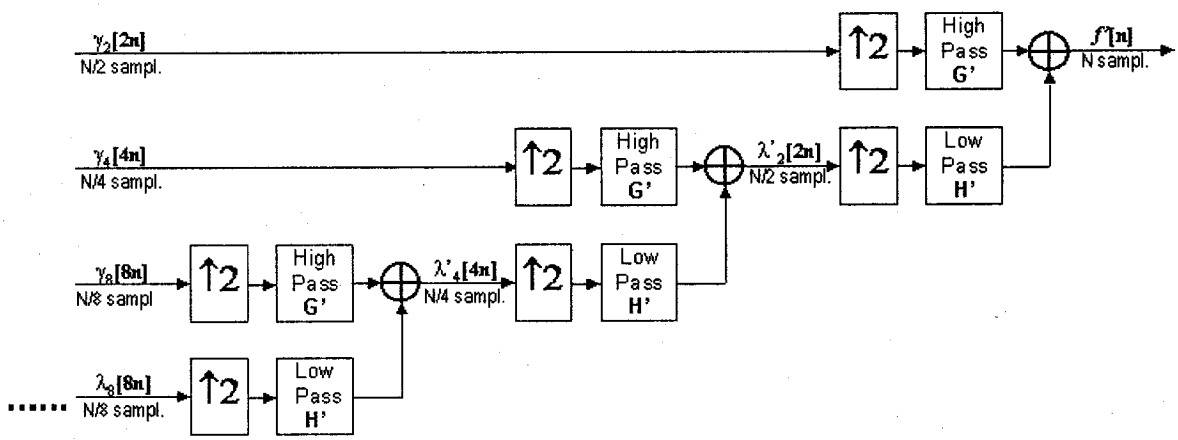

Figure 7. DWT Synthesis structure

Theoretically, the reconstruction of the original signal requires ideal half band filters in both DWT analysis and DWT synthesis. Since the ideal half-band filters are not achievable in practice, several types of numerical filters that provide perfect reconstruction have been developed. The present thesis uses the Daubechies wavelet coefficients since they offer a minimal phase solution to the set of DWT non-linear equations. The method to achieve this solution has been developed by Ingrid Daubechies in [10].

2.10 Daubechies Wavelet Coefficients

The set of nonlinear equations that provide solutions for DWT coefficients is obtained from conditions imposed to the orthogonal matrix of DWT coefficients, and from the vanishing moments of DWT polynomial representation.

A four-tap filter structure is used to deduct the equations. The coefficients **h0**, **h1**, **h2**, and **h3** are used to characterize both low-pass and high-pass filters according to QMF relation (2.21). The matrix relation (2.24) represents the full set of equations describing one DWT analysis stage similar to the one presented in [27].

$$\begin{bmatrix} \lambda_0 \\ \gamma_0 \\ \lambda_2 \\ \gamma_2 \\ \lambda_4 \\ \gamma_4 \\ \lambda_6 \\ \gamma_6 \end{bmatrix} = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & & & & \\ h_3 & -h_2 & h_1 & -h_0 & & & & \\ & & h_0 & h_1 & h_2 & h_3 & & \\ & & h_3 & -h_2 & h_1 & -h_0 & & \\ & & & & h_0 & h_1 & h_2 & h_3 \\ & & & & h_3 & -h_2 & h_1 & -h_0 \\ h_2 & h_3 & & & & & h_0 & h_1 \\ h_1 & -h_0 & & & & & h_3 & -h_2 \end{bmatrix} \times \begin{bmatrix} f_{-3} \\ f_{-2} \\ f_{-1} \\ f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \quad \textbf{(2.24)}$$

The empty spaces are null elements and the short form of this representation is:

$$[W] = [H] \cdot [F] \qquad (2.25)$$

Note the wraparound boundary condition represented on the last two rows of the matrix **H** that are virtually multiplied with the vector $[f_5\ f_6\ f_{-1}\ f_0\ f_1\ f_2\ f_3\ f_4]$. The wavelet coefficients λ_{2n} are obtained multiplying each odd row of **[H]** with the input vector $[\mathbf{f_{2n-k}}]$. The scaling function coefficients γ_{2n} are obtained multiplying each even row of **[H]** with the input vector $[\mathbf{f_{2n-k}}]$.

Since the DWT filters are build from orthogonal bases, the synthesis is obtained from multiplying the analysis coefficients with the transposed or orthogonal version of matrix **[H]**:

$$\begin{bmatrix} f'_{-3} \\ f'_{-2} \\ f'_{-1} \\ f'_{0} \\ f'_{1} \\ f'_{2} \\ f'_{3} \\ f'_{4} \end{bmatrix} \begin{bmatrix} h_0 & h_3 & & & & & h_2 & h_1 \\ h_1 & -h_2 & & & & & h_3 & -h_0 \\ h_2 & h_1 & h_0 & h_3 & & & & \\ h_3 & -h_0 & h_1 & -h_2 & & & & \\ & & h_2 & h_1 & h_0 & h_3 & & \\ & & h_3 & -h_0 & h_1 & -h_2 & & \\ & & & & h_2 & h_1 & h_0 & h_3 \\ & & & & h_3 & -h_0 & h_1 & -h_2 \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \gamma_0 \\ \lambda_2 \\ \gamma_2 \\ \lambda_4 \\ \gamma_4 \\ \lambda_6 \\ \gamma_6 \end{bmatrix} \qquad (2.26)$$

The synthesis coefficients $\mathbf{f'_n}$ are marked as "prime" to differentiate them from the original coefficients $\mathbf{f_n}$. The short form of DWT synthesis matrix representation is:

$$[F'] = [H]_T \cdot [W] \qquad (2.27)$$

The wraparound boundary condition appears now in the first two rows of the matrix $[H]_T$ that are virtually multiplied with the vector $[\lambda_0\ \gamma_0\ \lambda_2\ \gamma_2\ \lambda_4\ \gamma_4\ \lambda_{-2}\ \gamma_{-2}]$.

Perfect reconstruction ($\mathbf{f'_n = f_n}$) of the original input signal is achieved when the product between the analysis matrix **[H]** and its transposed version $\mathbf{[H]_T}$ is the unity matrix:

$$\mathbf{[H]\cdot[H]_T = [1]} \qquad \mathbf{(2.28)}$$

Two equations result from condition (2.28):

$$\mathbf{h_0^2 + h_1^2 + h_2^2 + h_3^2 = 2} \quad \mathbf{(2.29)}$$

$$\mathbf{h_0 \cdot h_2 + h_1 \cdot h_3 = 0} \qquad \mathbf{(2.30)}$$

Both high-pass **G** and low-pass **H** are FIR filters (see 2.7). The **z**-transform of a FIR filter **H** is a Laurent polynomial [12] of degree L-1:

$$\mathbf{H(z) = \sum_k h_k \cdot z^{-k},\ \ k=N..N+L-1} \quad \mathbf{(2.31)}$$

where **N** is a non-negative integer number.

For the particular case of four-tap filter used, the Laurent polynomial is:

$$\mathbf{H(z) = h_0 + h_1 \cdot z^{-1} + h_2 \cdot z^{-2} + h_3 \cdot z^{-3}} \qquad \mathbf{(2.32)}$$

Replacing the h_k coefficients with their QMF correspondents g_k in (2.31) results:

$$\mathbf{G(z) = h_3 - h_2 \cdot z^{-1} + h_1 \cdot z^{-2} - h_0 \cdot z^{-3}} \qquad \mathbf{(2.33)}$$

The expansion of four-tap DWT into Taylor series at $\mathbf{n = 0}$ using the Laurent polynomial representation, gives two more conditions corresponding to the vanishing moments of zero and first order:

$$h_0 - h_1 + h_2 - h_3 = 0 \qquad (2.34)$$

$$-1h_1 + 2h_2 - 3h_3 = 0 \qquad (2.35)$$

The equations (2.29), (2.30), (2.34), and (2.35) form a non-linear system that has many possible solutions. Ingrid Daubechies provided in [8] a minimal phase solution to this system, as follows:

$$h_0 = (1+3^{1/2})/(4\,2^{1/2}), \qquad h_1 = (3+3^{1/2})/(4\,2^{1/2}),$$

$$h_2 = (3-3^{1/2})/(4\,2^{1/2}), \qquad h_3 = (1-3^{1/2})/(4\,2^{1/2}).$$

Chapter 3 Architecture Description. The Modified Folded Wavelet

Two architectures have been implemented within this thesis: DWT Analysis and DWT Synthesis. Both architectures are using an improved version of *the folded structure* described in [15] and [24], called the *Modified Folded Wavelet*. This thesis presents also an improved *design method* based on a detailed tabular description of the architecture.

A three-stage DWT structure has been considered for both analysis and synthesis. According to the algorithm presented by K.K. Pahri and T. Nishitani in [24], the computations for a DWT structure having **N** stages are periodic in $\mathbf{M} = \mathbf{2}^{\mathbf{N}}$. This yields identical sets of the three-level DWT computations separated by a time index of eight. The index 8 is used to build the entire tabular structure of the Modified Folded Wavelet presented in this chapter.

3.1 DWT Analysis Architecture

The three-stage DWT analysis structure described in Figure 8 is implemented using four-tap FIR filters according to the characteristics described in Chapter 2, paragraphs 2.7 and 2.8. The DWT Analysis performs a multi-stage *dyadic high-pass-low-pass filtering* and the result is a collection of signals grouped in frequency sub-bands. The recursive structure of the algorithm allows the use of a single pair of filters to build the multi-stage DWT in piped-line architecture. Each filter is implemented using a Fast Processing Element (FPE) that performs four "multiply and accumulate" operations to evaluate one coefficient.

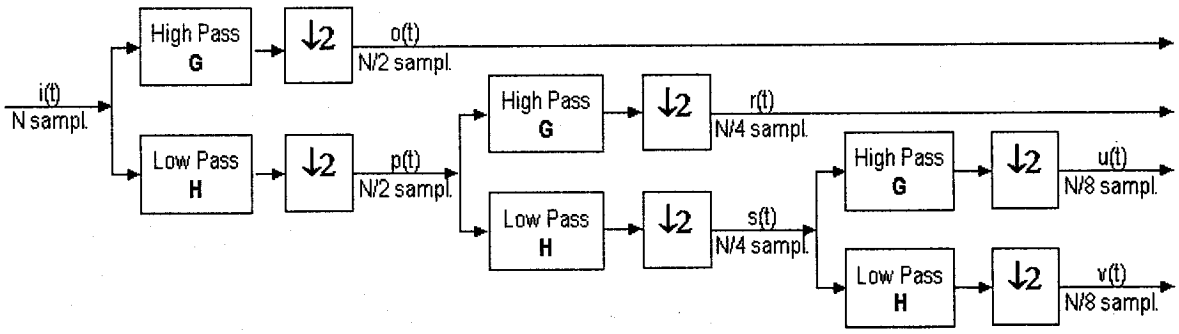

Figure 8. Three-level DWT Analysis Structure

The high-pass filter **G** and the low-pass filter **H** use respectively the same set of coefficients for all stages. For a set of **N** input coefficients, the stage **K** of each filter yields $N/2^K$ output coefficients. The recursive structure of both filters is given by formulas (2.18) and (2.19) in Chapter 2 paragraph 2.7. According to these formulas, one period of computation for the DWT coefficients described in Figure 6, is given by the following three groups of equations:

```
o(0)= i(0)g(0)+i(-1)g(1)+i(-2)g(2)+i(-3)g(3)  (3.1a)
o(2)= i(2)g(0)+i( 1)g(1)+i( 0)g(2)+i(-1)g(3)  (3.1b)
o(4)= i(4)g(0)+i( 3)g(1)+i( 2)g(2)+i( 1)g(3)  (3.1c)
o(6)= i(6)g(0)+i( 5)g(1)+i( 4)g(2)+i( 3)g(3)  (3.1d)
p(0)= i(0)h(0)+i(-1)h(1)+i(-2)h(2)+i(-3)h(3)  (3.1e)
p(2)= i(2)h(0)+i( 1)h(1)+i( 0)h(2)+i(-1)h(3)  (3.1f)
p(4)= i(4)h(0)+i( 3)h(1)+i( 2)h(2)+i( 1)h(3)  (3.1g)
p(6)= i(6)h(0)+i( 5)h(1)+i( 4)h(2)+i( 3)h(3)  (3.1h)
r(0)= p(0)g(0)+p(-2)g(1)+p(-4)g(2)+p(-6)g(3)  (3.1i)
r(4)= p(4)g(0)+p( 2)g(1)+p( 0)g(2)+p(-2)g(3)  (3.1j)
s(0)= p(0)h(0)+p(-2)h(1)+p(-4)h(2)+p(-6)h(3)  (3.1k)
s(4)= p(4)h(0)+p( 2)h(1)+p( 0)h(2)+p(-2)h(3)  (3.1l)
u(0)= s(0)g(0)+s(-4)g(1)+s(-8)g(2)+s(-12)g(3) (3.1m)
v(0)= s(0)h(0)+s(-4)h(1)+s(-8)h(2)+s(-12)h(3) (3.1n)
```

The first group of equations (3.1a) to (3.1h) produces two sets of four coefficients according to the first DWT stage, the second group (3.1i) to (3.1l) produces two sets of two coefficients according to the second stage and the third group (3.1m) to (3.1n) produces two coefficients according to the last stage.

3.2 Design Method Contribution

The folded DWT architecture is built according to a newly developed set of tables described below. The method uses the computation index of 8 for both analysis and synthesis, as mentioned at the beginning of Chapter 3. The tables are built according to the set of equations (3.1a) to (3.1n) for DWT analysis, respectively (3.2a) to (3.2n) for DWT synthesis.

The first step represents the construction of the *Processing Schedule* including pairs of coefficients calculated simuoltaneously by the two Fast Processing Elements (see Table 1). The order is deducted from the set of equations (3.1a) to (3.1n) by analyzing in which "clock cycle" the input samples are available for the next stage. The "clock cycle" represents the time interval when the respective coefficient is calculated. Each coefficient is written in an output register at the end of its computational cycle.

Table 1. Processing Schedule for DWT Analysis

Clock Cycle	G Filter	H Filter
0	o(0)	p(0)
1	r(0)	s(0)
2	o(2)	p(2)
3	u(0)	v(0)
4	o(4)	p(4)
5	r(4)	s(4)
6	o(6)	p(6)
7	-	-

The *Output Sequence* (see Table 2) is derived from Table 1. In this case, the "clock cycle" represents the time interval when the coefficient is available in the output register. Table 2 has ten rows to fully illustrate the output sequence. The default size for both Table 1 and Table 2 should be equal to the computation index that is eight in the present example. The minimum delay of one clock between the start of the processing and the first coefficient available at the output determines the improvement of the architecture presented in [21]. This improvement together with other optimizations resulting from the tabular method described in this paragraph, provide a 20% reduction of the hardware required to implement the DWT architecture.

Table 2. Output Sequence for DWT Analysis

Clock Cycle	Output coefficient
1	o(0)
2	r(0)
3	o(2)
4	u(0)
5	o(4)
6	r(4)
7	o(6)
8	v(0)
9	o(8)
10	r(8)

The *Coefficient Lifetime Chart* (Table 3) gathers all the data required to build the structure, and helps the designer to estimate the minimum number of registers needed to implement the *folded architecture* for both the input and the output coefficient sequence. The equations (3.1m) and (3.1n) indicate that both **u(0)** and **v(0)** calculated in cycle 3 of Table 1 require the coefficient **s(-12)** calculated 12 clock cycles before cycle 1 of the same table. This means that the *Coefficient Lifetime Chart* has to cover at least 12 + 8 = 20 clock cycles that will provide a valid description of the whole process. The *Coefficients processed* are copied from the *Processing Schedule* (Table 1) and the *Coefficients required* are deducted from the equations (3.1a)-(3.1n). The *Lifetime chart* is built inspecting *Coefficient processed* and placing each required coefficient, two clock cycles (one for processing and one for FPE output register) below its initial processing time shown in *Coefficients required* (see Table 3). The "end of life" of a coefficient corresponds to its last occurrence in *Coefficients required* column.

Table 3. Coefficient lifetime chart for DWT Analysis.

Cycle	Coefficients processed	Coefficients required	Coefficient lifetime chart							Live variables
0	o(0) p(0)	i(0) I(-1) i(-2) i(-3)								0
1	r(0) s(0)	p(0) p(-2) p(-4) p(-6)								0
2	o(2) p(2)	i(2) I(1) i(0) i(-1)	p(0)							1
3	u(0) v(0)	s(0) s(-4) s(-8) s(-12)		s(0)						2
4	o(4) p(4)	i(4) I(3) i(2) i(1)			p(2)					3
5	r(4) s(4)	p(4) p(2) p(0) p(-2)	p(0)			v(0)				4
6	o(6) p(6)	i(6) I(5) i(4) i(3)					p(4)			4
7	-	-						s(4)		5
8	o(8) p(8)	i(8) I(7) i(6) i(5)				v(0)			p(6)	6+0=6
9	r(8) s(8)	p(8) p(6) p(4) p(2)			p(2)		p(4)			5+0=5
10	o(10) p(10)	i(10) i(9) i(8) i(7)								3+1=4
11	u(8) v(8)	s(8) s(4) s(0) s(-4)		s(0)						3+2=5
12	o(12) p(12)	i(12) i(11) i(10) i(9)								2+3=5
13	r(12) s(12)	p(12) p(10) p(8) p(6)							p(6)	2+4=6
14	o(14) p(14)	i(14) i(13) i(12) i(11)								1+4=5
15	-	-								1+5=6
16	o(16) p(16)	i(16) i(15) i(14) i(13)								1+6+0=7
17	r(16) s(16)	p(16) p(14) p(12) p(10)								1+5+0=6
18	o(18) p(18)	i(18) i(17) i(16) i(15)								1+3+1=5
19	u(16) v(16)	s(16) s(12) s(8) s(4)						s(4)		1+3+2=6
20	o(20) p(20)	i(20) i(19) i(18) i(17)								0+2+3=5

The total number of *Live variables* is calculated by adding the active coefficients contained in every row of the table. The sequence starting in *cycle 0* of Table 3 is 20 clock cycles long and repeats every 8 cycles, resulting a maximum overlap of three sequences. The maximum number of *Live variables* resulting from adding the overlapped sequences is seven in this case.

The next step requires the construction of two *Forward-Backward Register Allocation* (FBRA) tables: one for the input and another for the output. Since the DWT analysis input structure is easy to implement with three registers connected in a chain, only the *Forward-Backward Register Allocation* for the output is described in Table 4.

Table 4. Forward-Backward Register Allocation for DWT Analysis output.

Cycle	H_out	R1	R2	R3	R4	R5	R6	R7
1	p(0)		p(-2)	s(-4)	p(-4)	s(-12)	p(-6)	s(-8)
2	s(0)	p(0)		p(-2)	s(-4)	s(-8)	s(-12)	
3	p(2)	s(0)	p(0)		p(-2)	s(-4)	s(-8)	s(-12)
4	v(0)	p(2)	s(0)	p(0)		p(-2)	s(-4)	
5	p(4)	v(0)	p(2)	s(0)	p(0)		p(-2)	s(-4)
6	s(4)	p(4)	v(0)	p(2)	s(0)	s(-4)		
7	p(6)	s(4)	p(4)	v(0)	p(2)	s(0)	s(-4)	
8		p(6)	s(4)	p(4)	v(0)	p(2)	s(0)	s(-4)
9	p(8)		p(6)	s(4)	p(4)	s(-4)	p(2)	s(0)
10	s(8)	p(8)		p(6)	s(4)	s(0)	s(-4)	
11	p(10)	s(8)	p(8)		p(6)	s(4)	s(0)	s(-4)
12	v(8)	p(10)	s(8)	p(8)		p(6)	s(4)	
13	p(12)	v(8)	p(10)	s(8)	p(8)		p(6)	s(4)
14	s(12)	p(12)	v(8)	p(10)	s(8)	s(4)		
15	p(14)	s(12)	p(12)	v(8)	p(10)	s(8)	s(4)	
16		p(14)	s(12)	p(12)	v(8)	p(10)	s(8)	s(4)
17	p(16)		p(14)	s(12)	p(12)	s(4)	p(10)	s(8)
18	s(16)	p(16)		p(14)	s(12)	s(8)	s(4)	
19	p(18)	s(16)	p(16)		p(14)	s(12)	s(8)	s(4)
20	v(16)	p(18)	s(16)	p(16)		p(14)	s(12)	
21	p(20)	v(16)	p(18)	s(16)	p(16)		p(14)	s(12)

This table is built from the *Lifetime chart* and includes the FPE output register **H_out** together with the seven registers required to preserve the maximum number of *Live variables* resulting from Table 3. Each coefficient present in the *Lifetime chart* of Table 3 is passed along a set of eight registers **H_out** and **R1** to **R7,** chained in a FIFO structure.

A special attention is given to the variables that have a life longer than eight cycles. Those have to be fed back in the structure to a suitable free register. The feedback operation is illustrated by arrows. The coefficients are underlined when they are used for the last time in the structure. The coefficients in shades of gray belong to adjacent computations

periods, and their index is obtained by adding or subtracting 8 from the base index written in black.

The last step may be skipped when the architecture can be easily implemented from the *Forward-Backward Register Allocation* table. The *Multiplexing Scheme* (Table 5) describes the multiplexing structure for all the selectable inputs resulting from the FBRA table. Only a number six multiplexers are required for this architecture yielding six rows for Table 5: four dedicated to FPE inputs, one for DWT analysis output, and one for **R5** input. Since the computation index is eight, the *Multiplexing Scheme* has eight columns, each column describing which input is fed to the respective multiplexer output.

The purpose of this table is to design a more efficient controller, identifying all possible redundant sequences existing in the proposed architecture. From this table, the designer can easily deduct the *modulo-k* repetitive sequences, or to take advantage of unused cycles for reduced selection schemes. For example, the "**8k+7**" cycle in Table 5 is considered as part of "**4k+3**" cycle. Table 5 provides the complete information to build the full DWT optimized structure described in Figure 9.

Table 5. Multiplexing Scheme for DWT Analysis.

Mux \ cycle	8k (2k,4k)	8k+1 (4k+1)	8k+2 (2k)	8k+3 (4k+3)	8k+4 (2k,4k)	8k+5 (4k+1)	8k+6 (2k)	8k+7 (4k+3)
MUX0	Input	H_out	Input	R1	Input	H_out	Input	X(R1)
MUX1	D1	R2	D1	R5	D1	R2	D1	X(R5)
MUX2	D2	R4	D2	R6	D2	R4	D2	X(R6)
MUX3	D3	R6	D3	R7	D3	R6	D3	X(R7)
InR5	R7	R7	R4	R4	R4	R7	R4	R4
Output	R4	G_out	G_out	G_out	G_out	G_out	G_out	G_out

The "don't care" selections starting with "X" are replaced by the selection in brackets written beside. The controller supplies selection signals for the multiplexer structure, using only the synchronous counter outputs **sel[1:0]**, and the *modulo-k* signals. The input selections for cycles (8k, 8k+2, 8k+4, 8k+6) have merged to 2k, cycles (8k+1, 8k+5) have merged to 4k+1, cycles (8k+3, 8k+7) have merged to 4k+3, and cycles (8k, 8k+4) have merged to 4k. The selection for InR5 (8k, 8k+1, 8k+5) is equivalent to (8k, 4k+1).

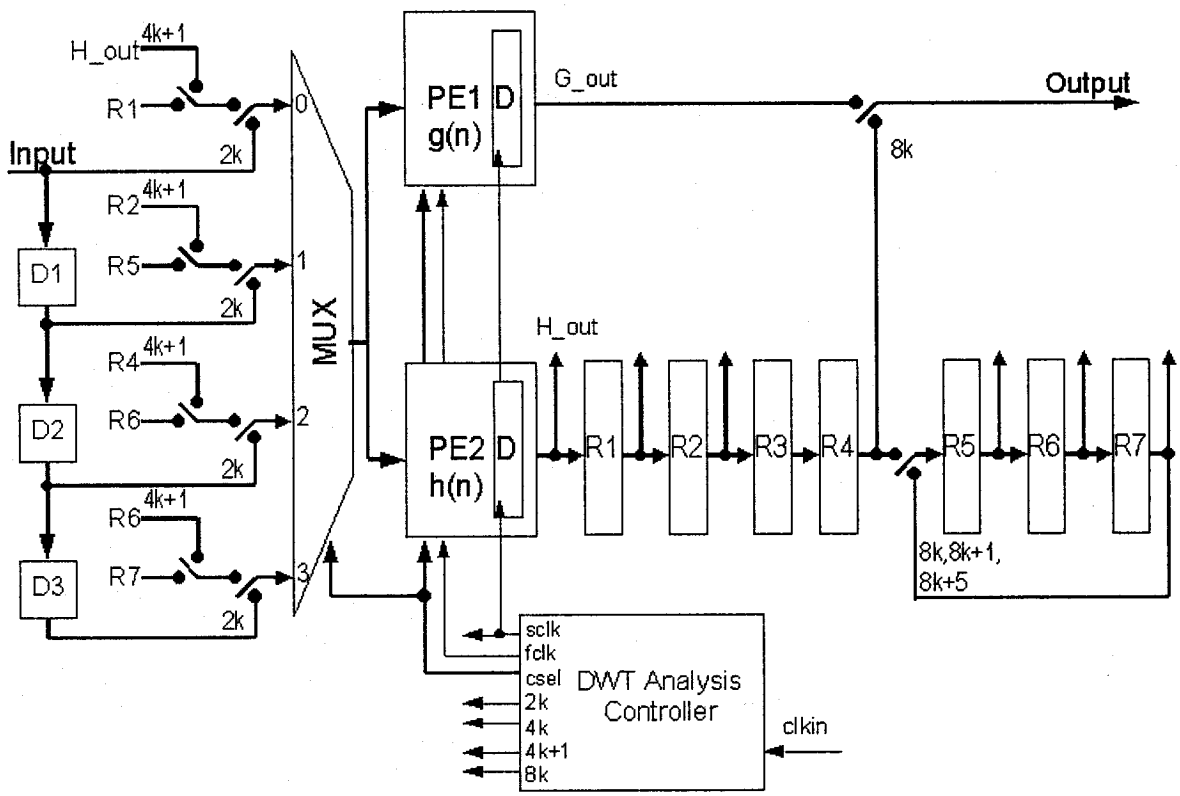

Figure 9. Folded DWT Analysis Architecture

The selections for InR5 and Output have to be active prior to the positive edge of the next clock cycle. Each label corresponds to the selection signal generated by the controller.

3.3 DWT Synthesis Architecture

The synthesis fully reconstructs the original signal when the filters are built from orthogonal basis according to Chapter 2, paragraphs 2.8 and 2.9. The three-stage DWT synthesis architecture described in Figure 10 is implemented as a modified version of the four-tap FIR filters used for DWT analysis. The recursive structure of synthesis allows also the use of a single pair of Fast Processing Elements to build the multi-stage DWT in piped-line architecture.

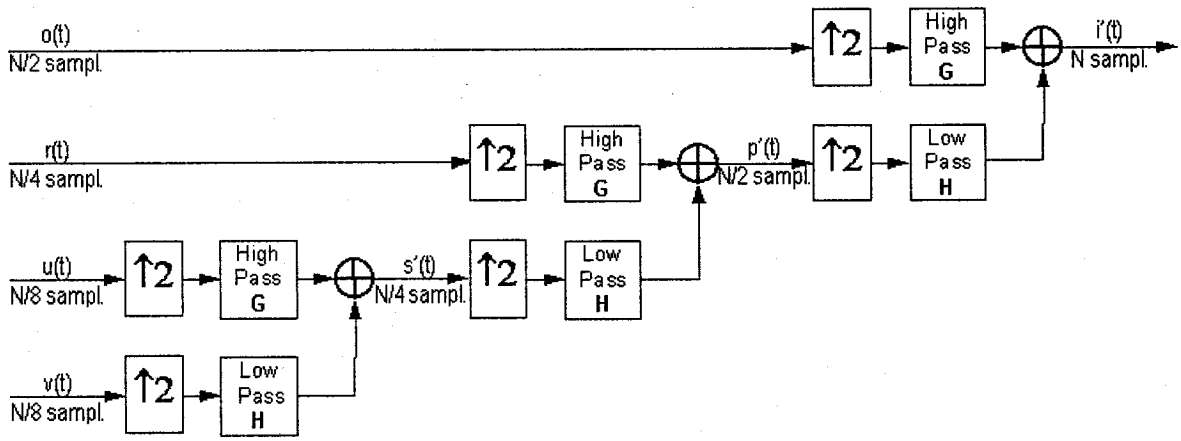

Figure 10. Three-level DWT Synthesis Structure

Both high-pass filter **G** and low-pass filter **H** use respectively the same set of coefficients as the analysis but in reversed order according to the relations (2.22) given in Chapter 2, paragraph 2.9. Each filter is preceded by an up-sampling stage that adds a null coefficient after each input coefficient. The recursive structure of the recombined filter is given by formula (2.23) in Chapter 2 paragraph 2.9. The up sampling with null coefficients reduces the synthesis equations from eight "multiply and accumulate" operations to only four. These four operations use two pair of even filter coefficients to calculate the even output samples and the other two pairs of odd filter coefficients to calculate the odd output

samples. The resulting simplified set of equations representing one period of computation for DWT synthesis is:

$$s'(0) = u(0)g(2)+u(-8)g(0)+ v(0)h(2)+ v(-8)h(0) \quad (3.2a)$$

$$s'(4) = u(0)g(3)+u(-8)g(1)+ v(0)h(3)+ v(-8)h(1) \quad (3.2b)$$

$$p'(0) = r(0)g(2)+r(-4)g(0)+s'(0)h(2)+s'(-4)h(0) \quad (3.2c)$$

$$p'(2) = r(0)g(3)+r(-4)g(1)+s'(0)h(3)+s'(-4)h(1) \quad (3.2d)$$

$$p'(4) = r(4)g(2)+r(0)g(0)+s'(4)h(2)+s'(0)h(0) \quad (3.2e)$$

$$p'(6) = r(4)g(3)+r(0)g(1)+s'(4)h(3)+s'(0)h(1) \quad (3.2f)$$

$$i'(0) = o(0)g(2)+o(-2)g(0)+p'(0)h(2)+p'(-2)h(0) \quad (3.2g)$$

$$i'(1) = o(0)g(3)+o(-2)g(1)+p'(0)h(3)+p'(-2)h(1) \quad (3.2h)$$

$$i'(2) = o(2)g(2)+o(0)g(0)+p'(2)h(2)+p'(0)h(0) \quad (3.2i)$$

$$i'(3) = o(2)g(3)+o(0)g(1)+p'(2)h(3)+p'(0)h(1) \quad (3.2j)$$

$$i'(4) = o(4)g(2)+o(2)g(0)+p'(4)h(2)+p'(2)h(0) \quad (3.2k)$$

$$i'(5) = o(4)g(3)+o(2)g(1)+p'(4)h(3)+p'(2)h(1) \quad (3.2l)$$

$$i'(6) = o(6)g(2)+o(4)g(0)+p'(6)h(2)+p'(4)h(0) \quad (3.2m)$$

$$i'(7) = o(6)g(3)+o(4)g(1)+p'(6)h(3)+p'(4)h(1) \quad (3.2n)$$

The first group of equations (3.2a) to (3.2b) produces a pair of coefficients every eight clock cycles according to the first synthesis stage, the second group (3.2c) to (3.2f) produces a pair of coefficients every four clock cycles according to the second stage, and the third group (3.1g) to (3.1n) produces a pair of coefficients every two clock cycles according to the last stage. The computation schedule for DWT synthesis has the same period as the analysis, thus it can be implemented in a similar manner.

According to paragraph 3.2, the first step is the construction of the *Processing Schedule* including pairs of coefficients calculated simuoltaneously by the two Fast Processing Elements. The order is deducted from the analysis output shedule (Table 2) and from the set of equations (3.2a) to (3.2n). In the following example, the DWT synthesis is designed

to decode directly the analysis output coefficients as they are produced. Since **v(0)** is available in the 8th cycle of the analysis (see Table 2), the computation of **s'(0)** and **s'(4)** can begin in odd filters and even filters in this cycle.

The *Processing Schedule* for one set of computations is shown in Table 6. In this schedule, the computations of **s'(i)** are spaced eight cycles apart, **p'(i)** are spaced four cycles apart , and **i'(i)** are spaced two cycles apart. Each "hyphen" represents a null operation. The "clock cycle" represents the time interval when the respective coefficient is calculated. Each coefficient is written in an output register at the end of its computational cycle.

Table 6. Processing Schedule for DWT Synthesis

Clock Cycle	Odd Filter	Even Filter
8	S'(0)	s'(4)
9	i'(-2)	i'(-1)
10	P'(0)	p'(2)
11	i'(0)	i'(1)
12		
13	i'(2)	i'(3)
14	P'(4)	p'(6)
15	i'(4)	i'(5)
16	S'(8)	s'(12)
17	i'(6)	i'(7)

The *Output Sequence* (Table 7) is derived from Table 6. In this case, the "clock cycle" represents the time interval when the coefficient is available in the output register. Both Table 6 and Table 7 have ten rows to fully illustrate the processing sequence. The default size for both Table 6 and Table 7 should be equal to the computation index that is eight for the synthesis structure.

Table 7. Output Sequence for DWT Synthesis

Clock Cycle	Output
10	i'(-2)
11	i'(-1)
12	i'(0)
13	i'(1)
14	i'(2)
15	i'(3)
16	i'(4)
17	i'(5)
18	i'(6)
19	i'(7)

Two *Coefficient Lifetime Charts* and two *Forward-Backward Register Allocation* tables are required for the DWT synthesis case, since both the input and the output structures are complex in this case. The *Coefficient Lifetime Chart for DWT synthesis input* (Table 8) gathers all the data required to build the input structure, and provide the minimum number of registers needed to implement the *folded architecture* for the input coefficient sequence. Similar to DWT analysis, the *Coefficient Lifetime Chart* has to cover at least 20 clock cycles to provide a valid description of the whole process. The *Coefficients processed* (Table 8) are copied from the *Processing Schedule* (Table 6) and the *Coefficients required* are deducted from the equations (3.2a)-(3.2n). The *Lifetime chart* (Table 8) is built inspecting *Coefficient processed* and placing each required coefficient one clock cycle below its occurrence in the DWT analysis output sequence shown in Table 2. The "end of life" of a coefficient corresponds to its last occurrence in *Coefficients required* (Table 8) column. Following the design method defined in 3.2, twelve input registers are required for the structure described in Table 8.

Table 8. Coefficient lifetime chart for DWT Synthesis input registers.

Cycle	Coefficient processed	Coefficient needed	Coefficient lifetime chart								Live variables
1	i'(-10) i'(-9)	o(-10) o(-12) p'(-10) p'(-12)									0
2	p'(-8) p'(-6)	r(-8) r(-12) s'(-8) s'(-12)	o(0)								1
3	i'(-8) i'(-7)	o(-8) o(-10) p'(-8) p'(-10)		r(0)							2
4	-	-			o(2)						3
5	i'(-6) i'(-5)	o(-6) o(-8) p'(-6) p'(-8)				u(0)					4
6	p'(-4) p'(-2)	r(-4) r(-8) s'(-4) s'(-8)					o(4)				5
7	i'(-4) i'(-3)	o(-4) o(-6) p'(-4) p'(-6)						r(4)			6
8	s'(0) s'(4)	u(0) u(-8) v(0) v(-8)							o(6)		7
9	i'(-2) i'(-1)	o(-2) o(-4) p'(-2) p'(-4)								v(0)	8
10	p'(0) p'(2)	r(0) r(-4) s'(0) s'(-4)									8+1=9
11	i'(0) i'(1)	o(0) o(-2) p'(0) p'(-2)									8+2=10
12	-	-									8+3=11
13	i'(2) i'(3)	o(2) o(0) p'(2) p'(0)	o(0)								8+4=12
14	p'(4) p'(6)	r(4) r(0) s'(4) s'(0)		r(0)							7+5=12
15	i'(4) i'(5)	o(4) o(2) p'(4) p'(2)			o(2)						6+6=12
16	s'(8) s'(12)	u(8) u(0) v(8) v(0)				u(0)				v(0)	5+7=12
17	i'(6) i'(7)	o(6) o(4) p'(6) p'(4)					o(4)				3+8=11
18	p'(8) p'(10)	r(8) r(4) s'(8) s'(4)						r(4)			2+8+1=11
19	i'(8) i'(9)	o(8) o(6) p'(8) p'(6)							o(6)		1+8+2=11
20	-	-									0+8+3=11

The construction of *Forward-Backward Register Allocation* for DWT input is deducted from the *Lifetime chart* of Table 8 and includes the DWT analysis output register as **Input** together with the twelve registers required to preserve the maximum number of *Live variables*.

The coefficients are underlined when they are used for the last time in the structure. The coefficients in shades of gray belong to adjacent computations periods, and their index is obtained by adding or subtracting 8 from the base index written in black.

Table 9. Forward-backward Register Allocation for DWT Synthesis input.

Cycle	Input	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
1	o(0)	v(-8)	o(-2)	r(-4)	o(-4)	u(-8)	o(-6)	r(-8)	o(-8)		o(-10)	r(-12)	o(-12)
2	r(0)	o(0)	v(-8)	o(-2)	r(-4)	o(-4)	u(-8)	o(-6)	r(-8)	o(-8)		o(-10)	r(-12)
3	o(2)	r(0)	o(0)	v(-8)	o(-2)	r(-4)	o(-4)	u(-8)	o(-6)	r(-8)	o(-8)		o(-10)
4	u(0)	o(2)	r(0)	o(0)	v(-8)	o(-2)	r(-4)	o(-4)	u(-8)	o(-6)	r(-8)	o(-8)	
5	o(4)	u(0)	o(2)	r(0)	o(0)	v(-8)	o(-2)	r(-4)	o(-4)	u(-8)	o(-6)	r(-8)	o(-8)
6	r(4)	o(4)	u(0)	o(2)	r(0)	o(0)	v(-8)	o(-2)	r(-4)	o(-4)	u(-8)	o(-6)	r(-8)
7	o(6)	r(4)	o(4)	u(0)	o(2)	r(0)	o(0)	v(-8)	o(-2)	r(-4)	o(-4)	u(-8)	o(-6)
8	v(0)	o(6)	r(4)	o(4)	u(0)	o(2)	r(0)	o(0)	v(-8)	o(-2)	r(-4)	o(-4)	u(-8)
9	o(8)	v(0)	o(6)	r(4)	o(4)	u(0)	o(2)	r(0)	o(0)		o(-2)	r(-4)	o(-4)
10	r(8)	o(8)	v(0)	o(6)	r(4)	o(4)	u(0)	o(2)	r(0)	o(0)		o(-2)	r(-4)
11	o(10)	r(8)	o(8)	v(0)	o(6)	r(4)	o(4)	u(0)	o(2)	r(0)	o(0)		o(-2)
12	u(8)	o(10)	r(8)	o(8)	v(0)	o(6)	r(4)	o(4)	u(0)	o(2)	r(0)	o(0)	
13	o(12)	u(8)	o(10)	r(8)	o(8)	v(0)	o(6)	r(4)	o(4)	u(0)	o(2)	r(0)	o(0)
14	r(12)	o(12)	u(8)	o(10)	r(8)	o(8)	v(0)	o(6)	r(4)	o(4)	u(0)	o(2)	r(0)
15	o(14)	r(12)	o(12)	u(8)	o(10)	r(8)	o(8)	v(0)	o(6)	r(4)	o(4)	u(0)	o(2)
16	v(8)	o(14)	r(12)	o(12)	u(8)	o(10)	r(8)	o(8)	v(0)	o(6)	r(4)	o(4)	u(0)
17	o(16)	v(8)	o(14)	r(12)	o(12)	u(8)	o(10)	r(8)	o(8)		o(6)	r(4)	o(4)
18	r(16)	o(16)	v(8)	o(14)	r(12)	o(12)	u(8)	o(10)	r(8)	o(8)		o(6)	r(4)
19	o(18)	r(16)	o(16)	v(8)	o(14)	r(12)	o(12)	u(8)	o(10)	r(8)	o(8)		o(6)
20	u(16)	o(18)	r(16)	o(16)	v(8)	o(14)	r(12)	o(12)	u(8)	o(10)	r(8)	o(8)	

The *Coefficient Lifetime Chart for DWT synthesis output* (Table 10) gathers all the data required to build the output structure, and provide the minimum number of registers needed to implement the *folded architecture* for the output coefficient sequence. The equations (3.2a) and (3.2b) indicate that both **s'(0)** and **s'(4)** calculated in cycle 8 of Table 6 require the coefficients **u(-8)** and **v(-8)** calculated 8 clock cycles before. This means that the *Coefficient Lifetime Chart* (Table 10) has to cover at least 8 + 8 = 16 clock cycles that will provide a valid description of the whole process. The *Coefficients processed* (Table 10) are copied from the *Processing Schedule* (Table 6) and the *Coefficients required* are deducted from the equations (3.2a)-(3.2n). The *Lifetime chart* (Table 10) is built inspecting *Coefficient processed* and placing each required coefficient, two clock cycles (one for processing and one for FPE output register) below its initial processing time shown in

Coefficients required column. The "end of life" of a coefficient corresponds to its last occurrence in *Coefficients required* column. The DWT synthesis output requires only five output registers according to Table 10.

Table 10. Coefficient lifetime chart for DWT Synthesis output registers.

Cycle	Coefficient processed	Coefficient needed	Coefficient lifetime chart							Live variables
8	s'(0) s'(4)	u(0) u(-8) v(0) v(-8)								0
9	i'(-2) i'(-1)	o(-2) o(-4) p'(-2) p'(-4)								0
10	p'(0) p'(2)	r(0) r(-4) s'(0) s'(-4)		s'(0)	s'(4)					2
11	i'(0) i'(1)	o(0) o(-2) p'(0) p'(-2)								2
12						p'(0)	p'(2)			4
13	i'(2) i'(3)	o(2) o(0) p'(2) p'(0)	i'(1)			p'(0)				5
14	p'(4) p'(6)	r(4) r(0) s'(4) s'(0)		s'(0)						3
15	i'(4) i'(5)	o(4) o(2) p'(4) p'(2)	i'(3)				p'(2)			3
16	s'(8) s'(12)	u(8) u(0) v(8) v(0)						p'(4)	p'(6)	3+0=3
17	i'(6) i'(7)	o(6) o(4) p'(6) p'(4)	i'(5)					p'(4)		4+0=4
18	p'(8) p'(10)	r(8) r(4) s'(8) s'(4)			s'(4)					2+2=4
19	i'(8) i'(9)	o(8) o(6) p'(8) p'(6)	i'(7)						p'(6)	2+2=4
20										0+4=4

The construction of *Forward-Backward Register Allocation* for DWT output is deducted from the *Lifetime chart* of Table 10 and includes the DWT synthesis output registers **O_odd** and **O_even** together with the five registers required to preserve the maximum number of *Live variables*. Both even and odd output coefficients are fed simultaneously to **R1** and **R2** registers during odd cycles.

A special attention is given to the variables that have a life longer than five cycles. Those have to be fed back in the structure to a suitable free register. The feedback operation is illustrated by arrows. The coefficients are underlined when they are used for the last time in the structure. The coefficients in shades of gray belong to adjacent computations

periods, and their index is obtained by adding or subtracting 8 from the base index written in black.

Table 11. Forward-backward Register Allocation for DWT Synthesis output.

Cycle	O_odd	O_even	R1	R2	R3	R4	R5
9	s'(4)	s'(0)	i'(-3)	p'(-2)	p'(-4)	s'(-4)	
10	i'(-1)	i'(-2)	s'(4)	s'(0)	p'(-2)		s'(-4)
11	p'(2)	p'(0)	i'(-1)	s'(4)	s'(0)	p'(-2)	
12	i'(1)	i'(0)	p'(2)	p'(0)	s'(4)	s'(0)	
13			i'(1)	p'(2)	p'(0)	s'(4)	s'(0)
14	i'(3)	i'(2)	s'(4)		p'(2)	s'(0)	
15	p'(6)	p'(4)	i'(3)	s'(4)		p'(2)	
16	i'(5)	i'(4)	p'(6)	p'(4)	s'(4)		
17	s'(12)	s'(8)	i'(5)	p'(6)	p'(4)	s'(4)	
18	i'(7)	i'(6)	s'(12)	s'(8)	p'(6)		s'(4)
19	p'(10)	p'(8)	i'(7)	s'(12)	s'(8)	p'(6)	
20	i'(9)	i'(8)	p'(10)	p'(8)	s'(12)	s'(8)	
21			i'(9)	p'(10)	p'(8)	s'(12)	s'(8)
22	i'(11)	i'(10)	s'(12)		p'(10)	s'(8)	
23	p'(14)	p'(12)	i'(11)	s'(12)		p'(10)	
24	i'(13)	i'(12)	p'(14)	p'(12)	s'(12)		
25	p'(18)	p'(16)	i'(13)	p'(14)	p'(12)	s'(12)	
26	i'(15)	i'(14)	p'(18)	p'(16)	p'(14)		s'(12)

The *Multiplexing Scheme for DWT synthesis* (Table 12) describes the multiplexing structure for all the selectable inputs resulting from the both FBRA tables corresponding to input (Table 9) and output (Table 11). A number of eight multiplexers are required for the synthesis, yielding eight rows for Table 12: four dedicated to FPE inputs, one for DWT synthesis output, and three for **R1, R2,** and **R4** input respectively. Since the computation index is eight, the *Multiplexing Scheme* has eight columns, each column describing which input is fed to the corresponding multiplexer output.

From Table 12, the designer can easily deduct the *modulo-k* repetitive sequences, or to take advantage of unused cycles for reduced selection schemes. The "don't care" selections starting with "X" are replaced by the selection in brackets written beside. The controller supplies selection signals for the multiplexer structure, using only the synchronous counter outputs **sel[1:0]**, and *modulo-k* signals. The input selections for cycles (8k, 8k+2, 8k+4, 8k+6) have merged to 2k, and the selections for cycles (8k+3, 8k+2) have merged to 4k+3.

Table 12. Multiplexing Scheme for DWT Synthesis.

Mux \ cycle	8k (2k)	8k+1	8k+2 (2k)	8k+3 (4k+3)	8k+4 (2k)	8k+5	8k+6 (2k)	8k+7 (4k+3)
MUX0	D4	D10	D8	D10	X(D8)	D10	D8	D10
MUX1	D12	D12	D12	D12	X(D12)	D12	D12	D12
MUX2	Input	R2	R2	O_even	X(R2)	R2	R1	O_even
MUX3	D8	R3	R5	R4	X(R4)	R3	R4	R4
InR1	O_odd	O_odd	O_odd	O_odd	O_odd	R4	O_odd	O_odd
InR2	R1	O_even	R1	O_even	R1	X(O_even)	R1	O_even
InR4	R3	X(R3)	R3	R3	R3	R5	R3	X(R3)
Output	O_even	R1	O_even	R1	O_even	R1	O_even	R1

Table 12 provides the complete information to build the full DWT optimized structure described in Figure 11. The selection of **MUX1** is unchanged during the whole synthesis sequence thus it can be replaced with a simple connection to register **D12.**

Many implementations are possible by changing the feedback path in Table 11. The present implementation was chosen due to its simplified multiplexing structure.

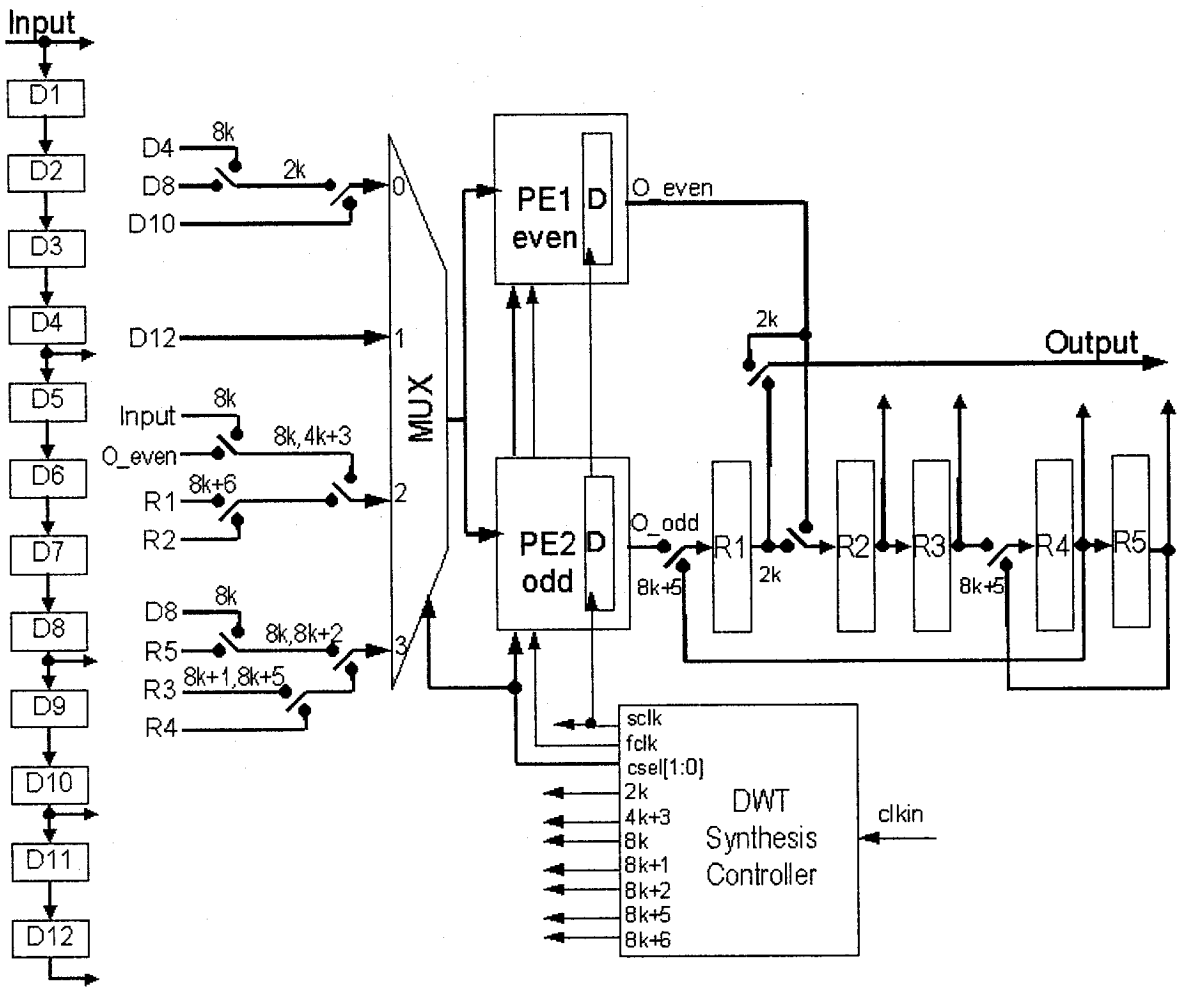

Figure 11. Folded Three-level DWT Synthesis Architecture

3.4 Fast Processing Element Contribution

Both DWT analysis and synthesis architectures include a pair of Fast Processing Elements. Two original Fast Processing Elements calculate the DWT coefficients given by equations (3.1a) to (3.1n) for analysis respectively (3.2a) to (3.2n) for synthesis. Each coefficient is calculated according to a four-tap FIR filter algorithm that requires four "multiply and accumulate" (MAC) operations for one output sample. This implies a MAC clock that is four times faster than the sample rate.

The MAC module is implemented using a fast and accurate processing element. The FPE uses an input multiplexer for filter coefficients synchronized with an external MUX (see Figure 9 and Figure 11) for DWT coefficients, and multiplies each correspondent pair with a fast Radix 2 Multiplier. The result of the multiplier is accumulated in a register with a fast Carry Select Adder. The high speed is achieved with a fixed-point implementation for both Radix 2 Multiplier and Carry Select Adder.

Since DWT perfect reconstruction is guaranteed for real numbers, the fixed-point representation yields computational noise due to coefficient truncation. A brief study of this noise based on experimental results is provided in paragraph 4.2.

The multiplier has been designed to process an 8-bit DWT coefficient together with a 25-bit filter coefficient, both represented as two's complement numbers. The decimal point for filter coefficients is placed two bits below MSB, one for the sign and the second for $2^0 = 1$. The maximum value represented on the 25-bit coefficient word is $\pm(2.0\text{-}2^{-23})$. The 32-bit partial result without sign compensation is further processed by the Carry Select Adder.

The multiplication of an 8-bit signed number with a 25-bit signed number is a 32-bit signed product. Special measures have to be considered in order to avoid arithmetical overflow and they are discussed in paragraph 3.6. Maximum precision is achieved during the multiply & accumulate process.

The result is truncated to 8 MSB in order to have the output word length matching the input. The filter clock "fclk" is synchronous with "sclk" and has the period four times smaller than "sclk".

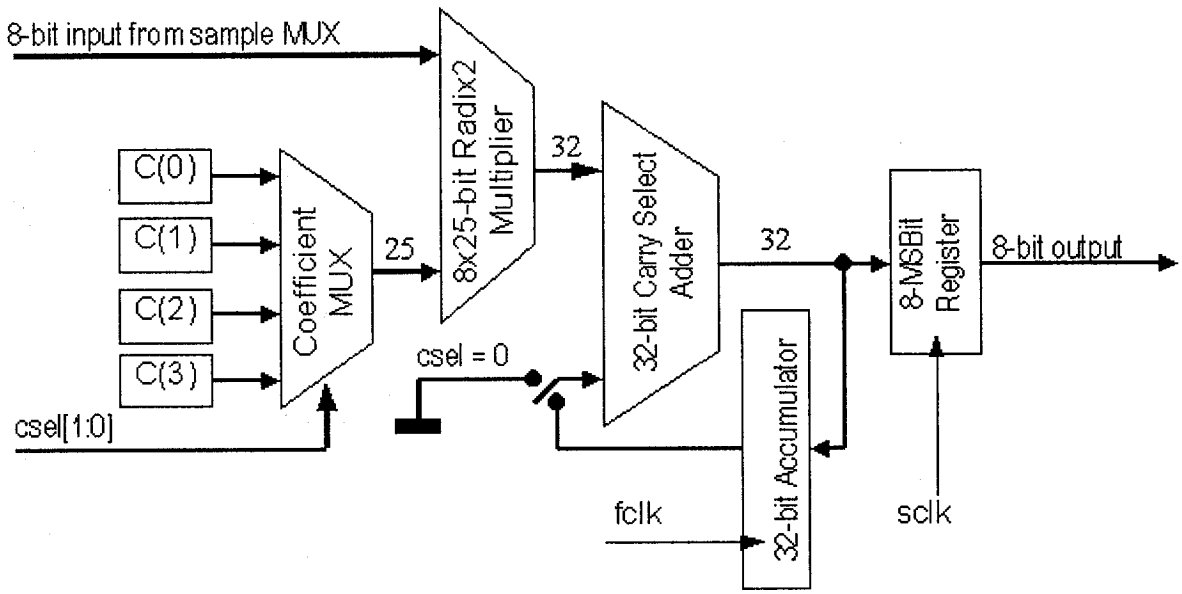

Figure 12. Fast Processing Element Structure

3.4.1 Radix2 Multiplier

The Radix2 Multiplier used within the FPE provides a fast fixed-point multiplication and a relative easy hardware implementation. The structure for this type of multiplier results from the following binary representation of two arbitrary integer numbers represented on **m** respectively **n** bits:

$$X = \sum_i x_i \cdot 2^i, \quad i=0..m-1, \quad Y = \sum_j y_j \cdot 2^j, \quad j=0..n-1 \qquad (3.3)$$

The multiplication of these numbers is

$$X \cdot Y = (\sum_i x_i \cdot 2^i) \cdot (\sum_j y_j \cdot 2^j) = \sum_i \sum_j (x_i \cdot y_j) \cdot 2^{i+j} \quad i=0..m-1, j=0..n-1 \quad (3.4)$$

Expression (3.4) can be rewritten as:

$$\mathbf{X \cdot Y = \sum_k p_k \cdot 2^k \quad k=0..m+n-1 \quad (3.5)}$$

where $\mathbf{p_k}$ represents the product $(\mathbf{x_i \cdot y_j})$ given by the binary function "AND".

The expression (3.5) is implemented in the example shown in Figure 13, where **m = 4** and **n = 3**. The structure has three groups of four multiplier elements (**ME**). First term bits $\mathbf{x_i}$ are fed in parallel to all the elements crossed by their correspondent oblique arrows, and second term bits $\mathbf{y_j}$ are fed in parallel to all the elements crossed by their correspondent horizontal arrows. The arrows crossing each multiplier's diagonal are carry bits propagated across the structure. The last line in the structure comprises four full adders that sum the most significant bits of the multiplication.

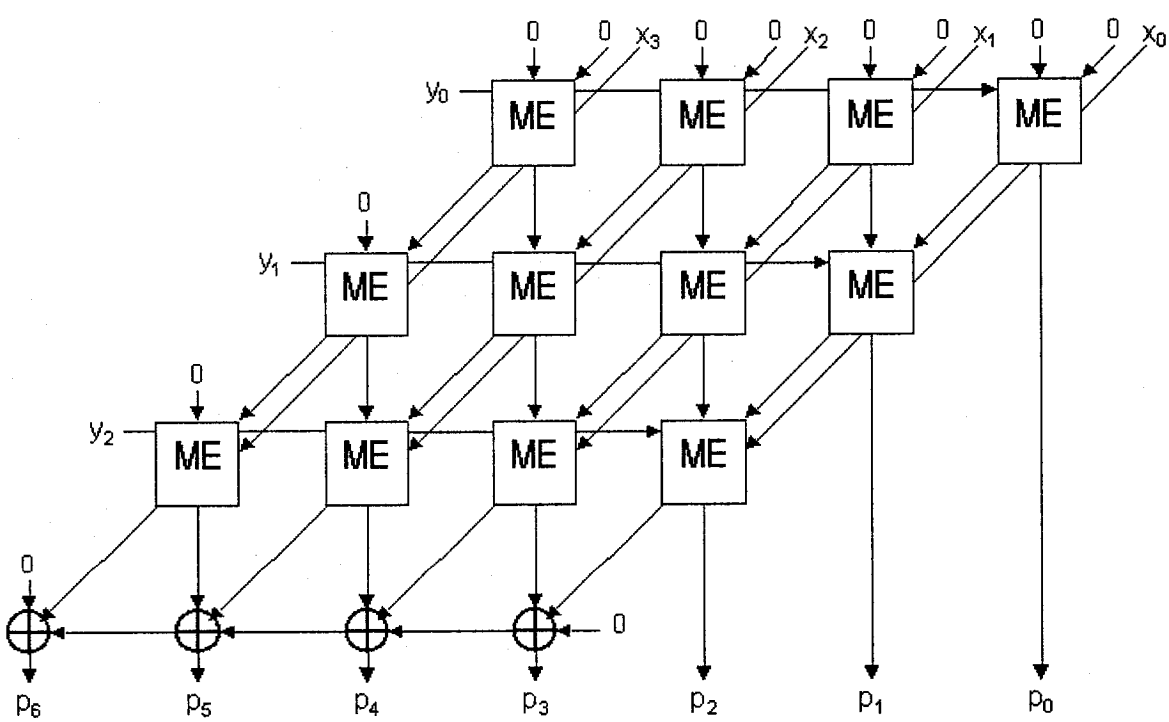

Figure 13. Radix2 Multiplier

The null bits fed in several points of the structure are provided for the completeness of the scheme. Each multiplier element include an "AND" gate and a full adder to calculate the partial product. The ME internal structure is presented in Figure 14.

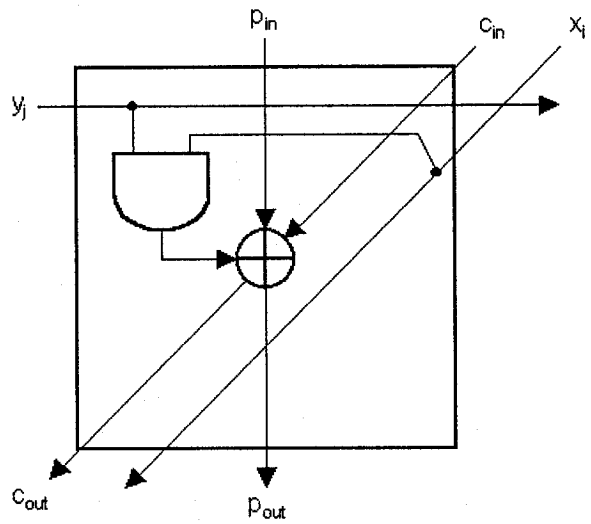

Figure 14. Multiplier Element

The previous stage product $\mathbf{p_{in}}$ is added to the actual product ($\mathbf{x_i \cdot y_j}$), together with the carry input $\mathbf{c_{in}}$, resulting the output product $\mathbf{p_{out}}$ and the carry output $\mathbf{c_{out}}$.

Negative numbers represented in two's complement are converted to positive numbers before multiplication. The combination of signs corresponding to each term determines the sign of the product. If the result is negative, the two's complement applies.

3.4.2 Carry Select Adder

A Carry Select Adder is implemented within the FPE to calculate the 32-bit accumulation. The carry input of this adder is used to accumulate two's complement numbers when the multiplication result is negative (see 3.4.1).

The Carry Select Adder reduces to half the time to propagate the carry signal through the whole structure. Since carry propagation is the major delay within an adder, this yields a reduction to half the time required for the whole addition.

The functionality of this type of fast adder is described in Figure 15 below. The algorithm can be applied recursively, resulting a further reduction of the addition time at the cost of increasing the number of gates used. The Fast Processing Element uses a 32-bit Carry Select Adder partitioned in groups of four-bit adder. The gate delay estimation shows that partitioning below four-bit is no longer efficient for any given IC technology.

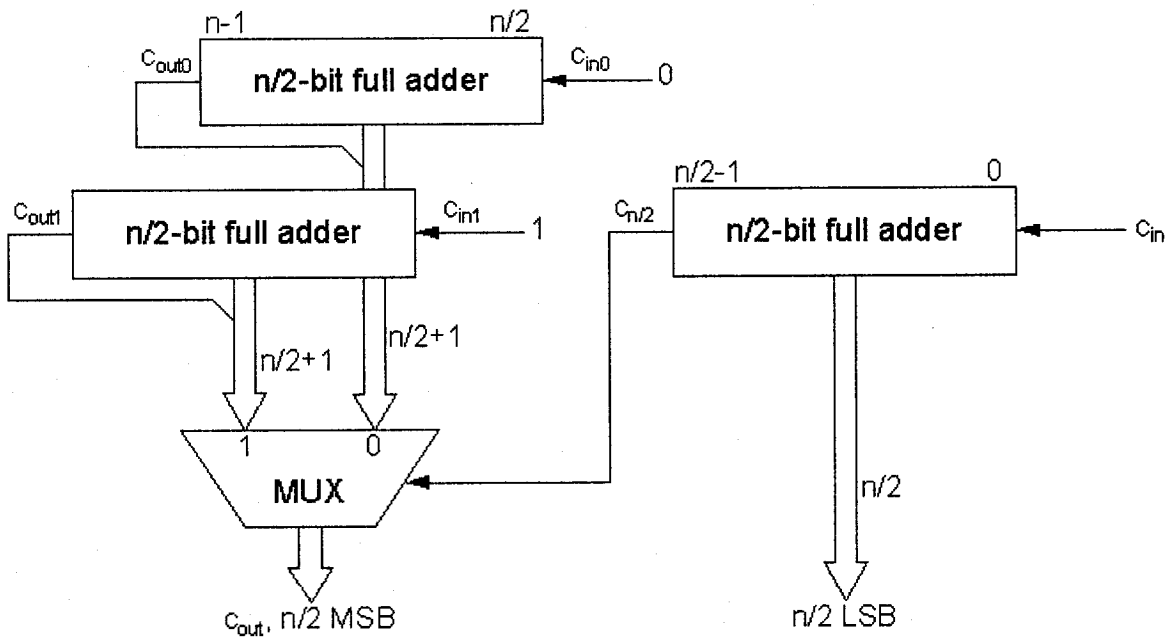

Figure 15. Carry Select Adder

The original **n**-bit adder is divided in two equal sections and **n** is an integer number representing a power of two. The first half representing the bits **0** to **n/2-1** is a full adder of length **n/2** and its carry out signal $\mathbf{c_{n/2}}$ is used to select the result for the second half of the adder representing the bits **n/2** to **n-1** together with the final carry out $\mathbf{c_{out}}$. The second part

of the adder comprises two **n/2**-bit full adders, the first one having the carry input set to "0" and the second one having the carry input set to"1". This structure allows the calculation in parallel for both possible carry input scenarios yielding a reduced addition time given by the formula:

$$\mathbf{T_{select\&add}(n) = T_{add}(n/2) + 1} \qquad \mathbf{(3.6)}$$

When the number of bits **n** is large, the addition time of the Carry Select Adder is reduced to half compared to the time of a normal full adder. The time in expression (3.6) is measured in gate delay units depending of the IC technology used.

The number of gates required to implement this structure is:

$$\mathbf{G_{select\&add}(n) = 3G_{add}(n/2) + G_{mux}} \qquad \mathbf{(3.7)}$$

where $\mathbf{G_{mux}}$ is the number of gates required to multiplex **n/2+1** bits. If $\mathbf{G_{mux}}$ is ignored, the Carry Select Adder requires **1½** times more hardware than a normal full adder.

3.5 Filter Coefficients

DWT similar to Fast Fourier Transform (FFT), turns a discrete vector of length N into a vector of N coefficients, which scale the transformation's base functions. The DWT is a linear operation that can be efficiently implemented as a recursion (see paragraphs 2.7 and 2.9). The requirement for the base transformation function to be orthogonal and reversible yields a set of constraining equations for discrete wavelets (see paragraph 2.10). Daubechies wavelet filter coefficients are the minimum phase solutions of this nonlinear equation set.

The DWT analysis algorithm given by equations (3.1a) to (3.1n) use the set of coefficients given in Table 13 and is based on Daubechies wavelets [11] of length 4. The DWT synthesis algorithm given by equations (3.2a) to (3.2n) use the same set of coefficients but the order is reversed according to paragraph

The transformation in 16-bit hexadecimal coefficients has been done according to the formulas:

$$hx_i = \mathbf{round}(2^{15} \times h_i) \qquad (3.8)$$

$$gx_i = (-1)^i \, \mathbf{round}(2^{15} \times h_{3-i}) \qquad (3.9)$$

Table 13. Daubechies coefficients.

Daubechies coefficients	Low-pass filter coefficients (hex)	High-pass filter coefficients (hex)
h0=0.48296291314	hx0 = 0x3DD2	gx0=0xEF70
h1=0.83651603074	hx1 = 0x6B13	gx1=0xE34F
h2=0.22414386804	hx2 = 0x1CB1	gx2=0x6B13
h3=-.12940952255	hx3 = 0xEF70	gx3=0xC22E

3.6 Fixed Point DWT Calculation Contribution

An original implementation of the fast signal-processing algorithm is described below. This implementation uses the fixed-point numerical representation. The major disadvantages of the fixed-point representation are the reduced precision, and the high probability to reach arithmetical overflow or underflow. These two deficiencies may introduce an important amount of noise in the processed signal.

Special measures have to be implemented to reduce the computational noise due to fixed point. The overflow is avoided by ensuring that the computations will never be greater than the number representing the maximum amplitude of the signal. A *novel* method that *normalizes* and *de-normalizes* the Daubechies coefficients is developed within this work to increase the precision of the computation without increasing the size of the operands.

Positive digital samples of **n** bits $s_i = [b_{n-1}, b_{n-2}, \ldots, b_0]$ are usually represented as sub-unitary fractions of the signal's maximum amplitude **A**, according to the relation:

$$s_i = A \cdot (\sum_k b_{n-k} \cdot 2^{-k}) \quad k=1..n \qquad (3.10)$$

where b_{n-1} is the most significant bit and the amplitude is normalized $A = 1$. If the signal is bipolar, one more bit is added before b_{n-1} to mark the sign, resulting an $n+1$ bit word. Signal processing should not produce samples greater than 2^n, corresponding to the maximum unity amplitude.

Analyzing the three-level DWT together with the four Daubechies coefficients, a DC signal having the maximum input amplitude $A_{imax} = 1$ will have the maximum coefficient amplitude A_{c3max} given by the formula:

$$A_{c3max} = (\sum_{k=0,3} h[k])^3 \cdot A_{imax} = 2^{3/2} \qquad (3.11)$$

The maximum coefficient amplitude A_{c3max} corresponds to the third stage low pass filter. The relation (3.11) shows that DWT analysis coefficients will have almost three times the amplitude of a DC or low frequency input signal.

The present thesis studies three solutions to this problem. The first one *divides* by 4 the input coefficients, decreasing the computation dynamic range to $\mathbf{A_{c3max}} = \mathbf{2^{-1/2}}$. The second solution *divides* by 2 the analysis coefficients, resulting the same dynamic range of $\mathbf{A_{c3max}} = \mathbf{2^{-1/2}}$. The third solution *divides* the coefficients by the sum $\mathbf{\sum_{k=0,3} h[k] = 2^{1/2}}$, resulting an optimized dynamical range of $\mathbf{A_{c3max}} = \mathbf{1}$ that yields the *most efficient arithmetical implementation* of the transform. It is important to mention that the second and the third solution require a *multiplication* of the corresponding synthesis coefficients with **2** respectively $\mathbf{2^{1/2}}$, to preserve the perfect reconstruction property according to paragraphs 2.10 and 3.5.

The high pass filter coefficients that result from the conjugated mirror filter structure according to paragraph 2.8, have an important property that produces null amplitude for DC input signal:

$$\mathbf{\sum_{k=0,3} g[k] = 0} \qquad \mathbf{(3.12)}$$

Since the image signal is always positive and DWT computation generates bipolar coefficients after the first high-pass filtering stage, the positive input signal has to be converted in bipolar two's complement samples before entering the DWT analysis. In this case, the DWT synthesis is followed by another conversion from two's complement to positive output signal. The conversion is simple, requiring only the negation of the most significant bit in both cases.

3.7 Two-dimension DWT

Several two-dimension DWT (2D-DWT) implementation methods are known from the literature. Some implementations perform the 2D DWT in both dimensions at the same time but these render a block structure for the processed image that may create visible pixel distortions in case of noisy image. This inconvenience is solved by starting to process all the rows and continuing with all the columns afterwards. The visible effect of image noise is reduced in this case.

The DWT architectures described in detail in paragraphs 3.1 and 3.3 are optimized to be implemented within a 2D DWT. A video or image memory of (**M x N**) pixels is required to calculate the DWT analysis coefficients for a monochrome image. For color images, three memories of (**M x N**) pixels are needed. Both **M** and **N** should be multiples of 8 to facilitate an accurate three-level DWT calculation.

The 2D processing is performed at the beginning row-by-row and afterwards column-by-column. The DWT coefficients are grouped in sub-bands starting with the *scaling function* corresponding to the last low-pass filter and, ending with the *wavelet* corresponding to the first high-pass filter. The 2D DWT image analysis is described in Figure 16.

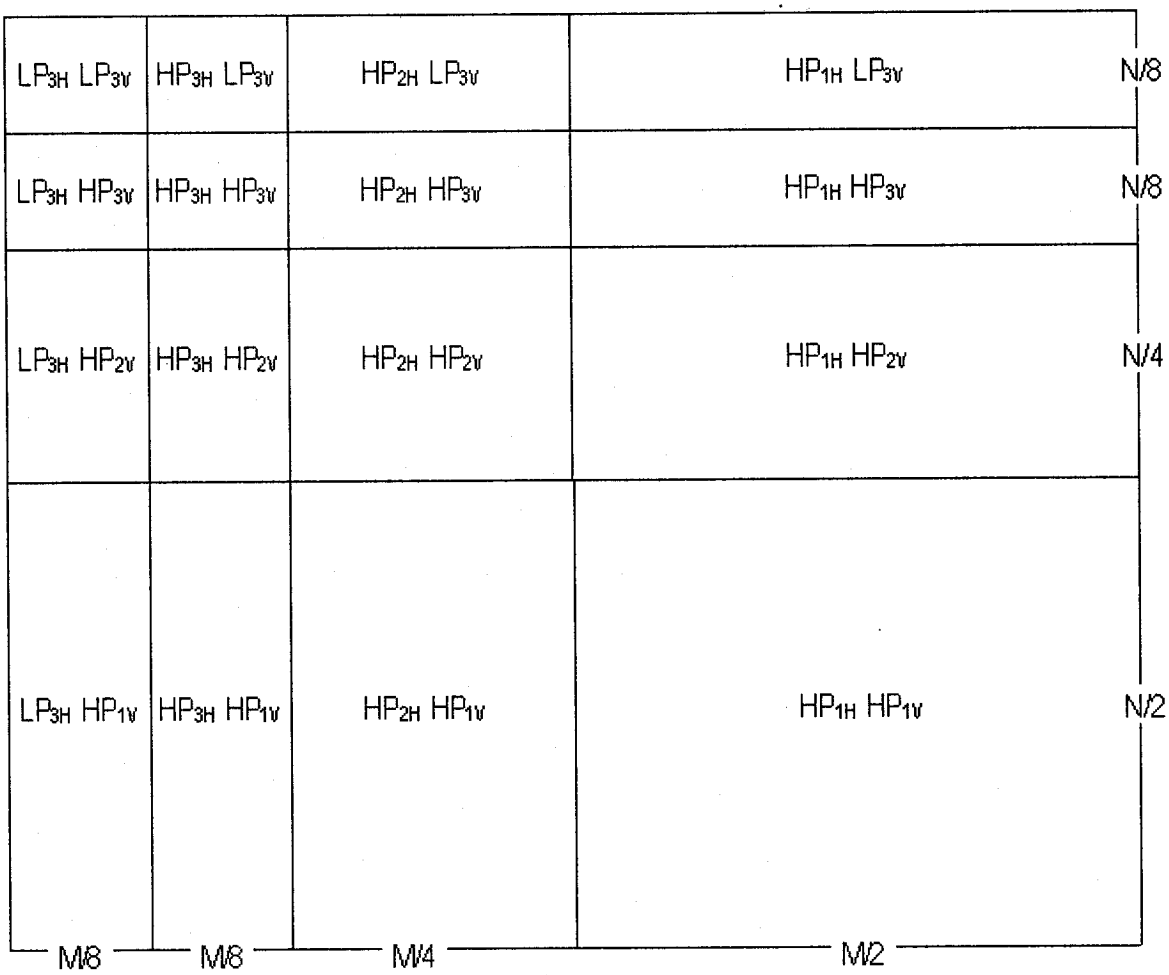

Figure 16. 2D DWT Image Analysis

The three level DWT requires a buffer of length **L = MAX (M, N)** to memorize each row or column before processing. During processing, the coefficients are written back in the original image memory in areas resulting from the dyadic division of image in M/2 to M/8 horizontal segments respectively N/2 to N/8 vertical segments.

The transformed image shown in Figure 16 has sixteen coefficient areas produced by combining four horizontal sub-bands with four vertical sub-bands. The upper-left corner of the image contains the most important information filtered by the last pair of low-pass filters $\mathbf{LP_{8H}\ LP_{8V}}$ corresponding to the scaling function. This area is usually left

unchanged by the compression algorithms. All other areas may have the pixel size truncated or even discarded, depending on the information they contain.

Chapter 4 Experimental Results and Contributions

4.1 Behavioral Models and HDL Code

The WaveLab software package available at http://www-stat.stanford.edu/~wavelab/ was used to perform behavioral simulations for DWT. This is a useful Wavelet toolkit developed in Matlab environment by specialists from Stanford University. It contains routines for DWT analysis and synthesis, and routines to calculate many types of DWT coefficients for different sizes of FIR filters, including Daubechies coefficients.

All hardware modules were coded using Verilog HDL. The code listings are given in Annex A. The following modules have been synthesized using a 0.35µm CMOS standard-cell library:

- *cs_adder.v*, 32-bit carry-select adder optimized for synthesis.
- *mult8x25.v*, 8-bit x 25-bit multiplier with two's complement capability;
- *FPE.v*, FPE module optimized for fixed point multiply & accumulate operations and implemented according to paragraph 3.4;
- *DWTcntl_A.v*, controller for DWT Analysis implemented with modulo-k counters;
- *DWTcntl_S.v*, controller for DWT Synthesis implemented with modulo-k counters;
- *DWT_top_A.v*, DWT Analysis module implemented according to Figure 9;
- *DWT_top_S.v*, DWT Synthesis module implemented according to Figure 11.

Several test-benches were designed in order to verify the functionality of each module:

- *t_cs_adder.v*, carry-select adder test yielding 2.462ns maximum delay in the synthesized structure.
- *t_mult8x25.v*, test for 8x25-bit multiplier yielding 6.1ns maximum delay in the synthesized structure;
- *t_DWTcntl_A.v*, test for DWT Analysis controller;
- *t_DWTcntl_S.v*, test for DWT Synthesis controller;
- *t_DWT_top.v*, test for the full-chain Analysis-Synthesis connected together.

4.2 Simulation Results

The test-bench *t_DWT_top.v* checks the functionality of both DWT Analysis and Synthesis connected together. This test uses three buffer memories, one for input samples, second for analysis coefficients, and the third for synthesis coefficients. These buffers are provided to facilitate the 2D DWT processing row by row then column by column as described in paragraph 3.7. A set of input files named "sinpXXXX.dat" was generated within *Matlab* environment to test the entire coding and decoding structure. Each file contains 8-bit samples of positive sinusoidal signal generated to test the arithmetical noise introduced by fixed point DWT calculation described in paragraph 3.6. The frequency of these signals was chosen to be in the low end of the DWT spectrum corresponding to the third stage low-pass filter, in order to highlight how the four DWT filters are working through attenuation.

The signal trace corresponding to DWT analysis output shown in the upper part of each waveform figure demonstrates the dyadic division in frequency sub-band. The low–pass

filter has the largest amplitude, and the amplitudes corresponding to the upper high-pass filters are attenuated in steps corresponding to their position in spectrum. The signal of the last two stages has negligible amplitude. The DWT synthesis traces shown in the lower part of each waveform figure, are a measure of the quality of the fixed point transform.

If compression is considered in this particular case of single tone waveform, the samples corresponding to all high-pass filters can be discarded without losing the quality of the reconstructed signal. This particular case also demonstrates that reducing the input samples to only four bits, an acceptable reconstruction is still possible.

The first waveform from Figure 17 represents the sinusoidal input signal used to demonstrate the advantages of the novel coefficient normalization proposed in paragraph 3.6. The normalized period of this signal is sixty-four times the sampling period and its frequency belongs to the third stage low-pass filter band.

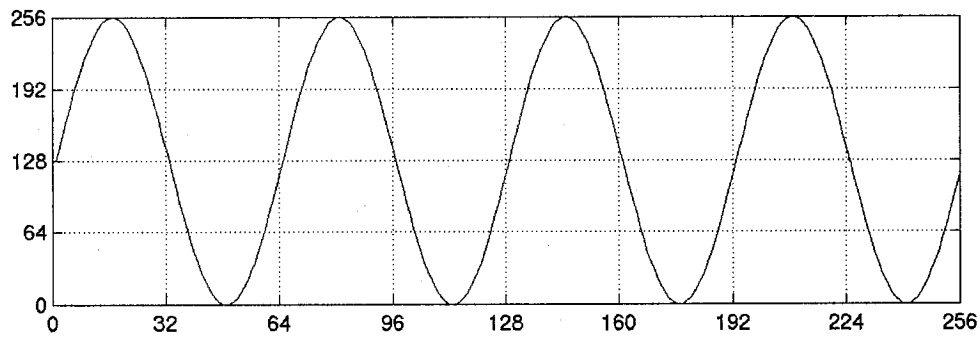

Figure 17. DWT Input Signal

The results for the three normalization cases proposed in paragraph 3.6, are presented below. The first case divides the amplitude of the input signal by 4 to avoid arithmetical overflow according to formula (3.11)

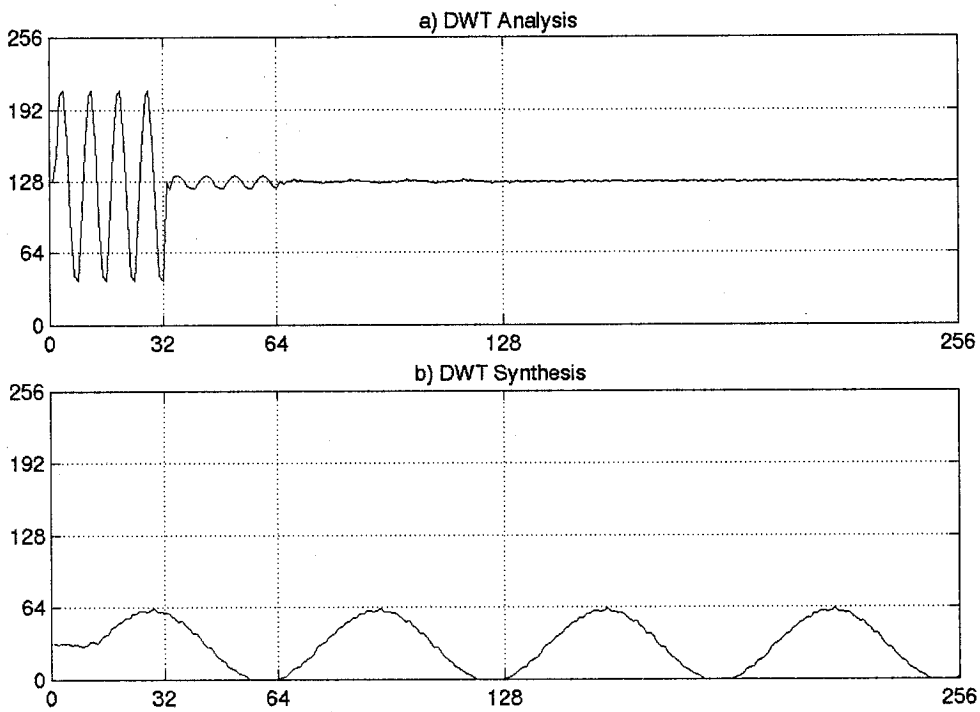

Figure 18. DWT Simulation with Input divided by 4

The coefficients resulting form the analysis have maximum peak-to-peak amplitude of 180 units that is 70% of the full-allowed range. The major disadvantage of this implementation is the small amplitude obtained at the synthesis output according to Figure 18 b). Since the computational noise due to sample and coefficient truncation is considered constant for any input signal, the signal to noise ratio at the synthesis output is reduced four times. If the synthesis output is multiplied by four to bring it back to its original amplitude, the noise is also amplified accordingly. The noisy waveform representing the multiplication by four is shown in Figure 19, and has a 38dB signal to noise ratio.

A better solution is to divide by 2 the Daubechies coefficients. The experimental results shown in are presented in Figure 20, and has a 42dB signal to noise ratio (SNR).

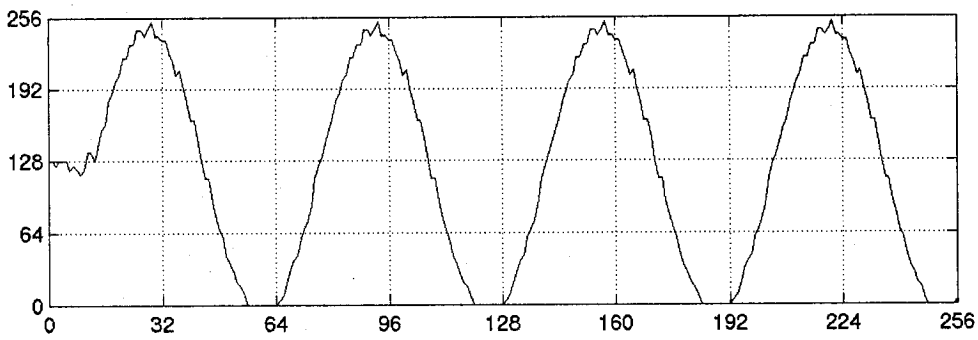

Figure 19. Divide by 4 Compensation for Synthesis output

The improvement noticed for the reconstructed signal from Figure 20 b) compared with the signal from Figure 19 is achieved due to an additional information bit used in the all the MAC operations. The proof of this statement is given by analyzing the multiplications that are performed using 5+24 = 29 bits without the sign for the first case compared to 7+23 = 30 for the second case. However, the second case leaves some room for improvement since the full multiplier range is 7+24 = 31 bits.

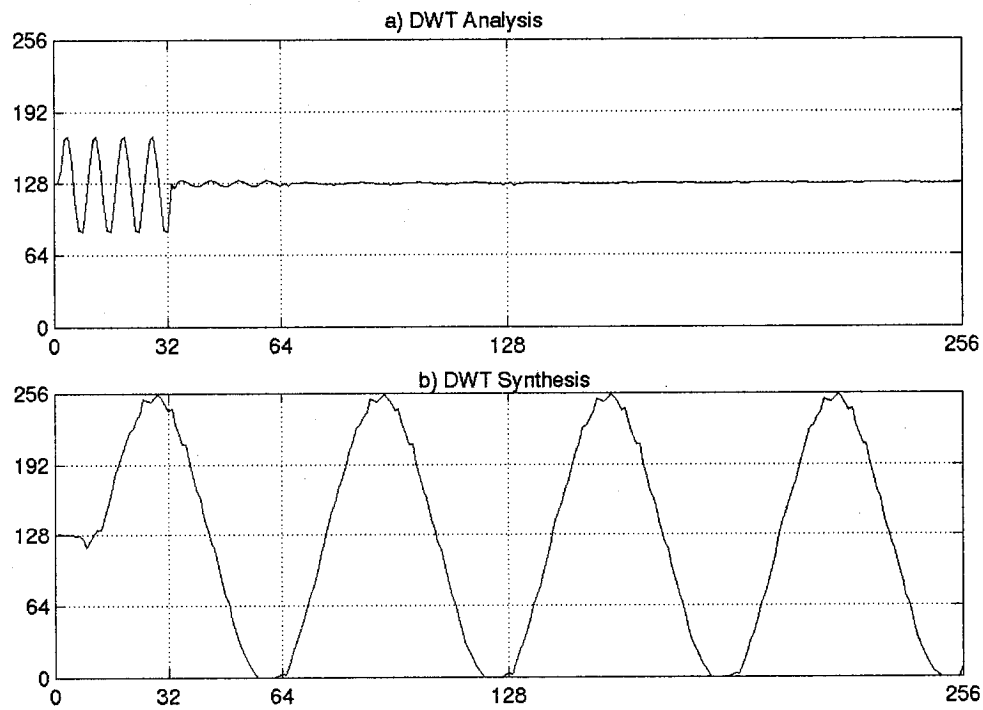

Figure 20. DWT Simulation with Coefficients Divided by 2

Also is important to highlight the reduced peak-to-peak amplitude of the analysis signal having only 90 units that represents 35.5% of the allowed range. This reduced amplitude for analysis coefficients will affect significantly the quality of the synthesis in case of losing information samples from the critical sub-bands, especially when the coded image signal is transmitted through a communication channel.

The third solution that normalizes the coefficients with the sum of the high pass filter utilizes both DWT structures at the maximum of their capability without producing arithmetic overflow. The results of the third solution simulation illustrate a full range achieved for the analysis in Figure 21 a) yielding a better 45dB SNR in Figure 21 b) compared to 42db SNR in Figure 20 b).

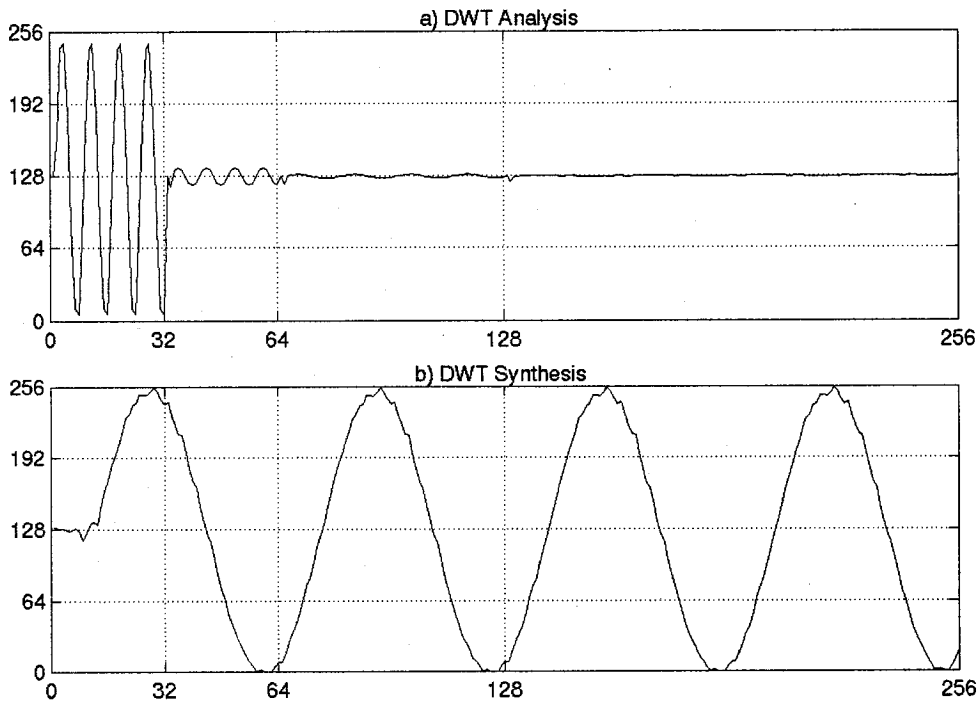

Figure 21. DWT Simulation with Coefficients divided by Σh_i

Again, the improvement is achieved through an additional information bit used in all the MAC operations. The proof of this statement is given by analyzing the multiplications that are performed using 7+23 = 30 bits without the sign for the second case compared to 7+24 = 31 for the third case.

The optimized structure obtained with the third proposed solution allows the use of smaller word size for Daubechies coefficients without deteriorating the overall quality of the transform. The results presented in Figure 22 were obtained with **17-bit** coefficients.

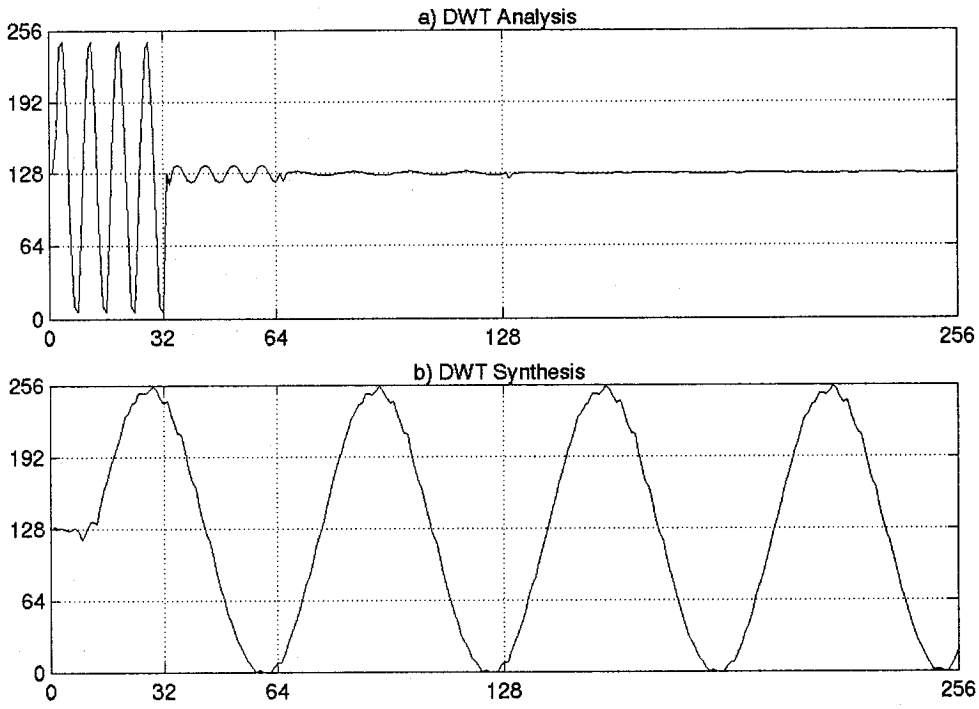

Figure 22. DWT Simulation with 17-bit Coefficients

Finally, a simulation using **9 bit** Daubechies coefficients has successfully concluded the entire optimization process yielding the same very good quality transform shown in .

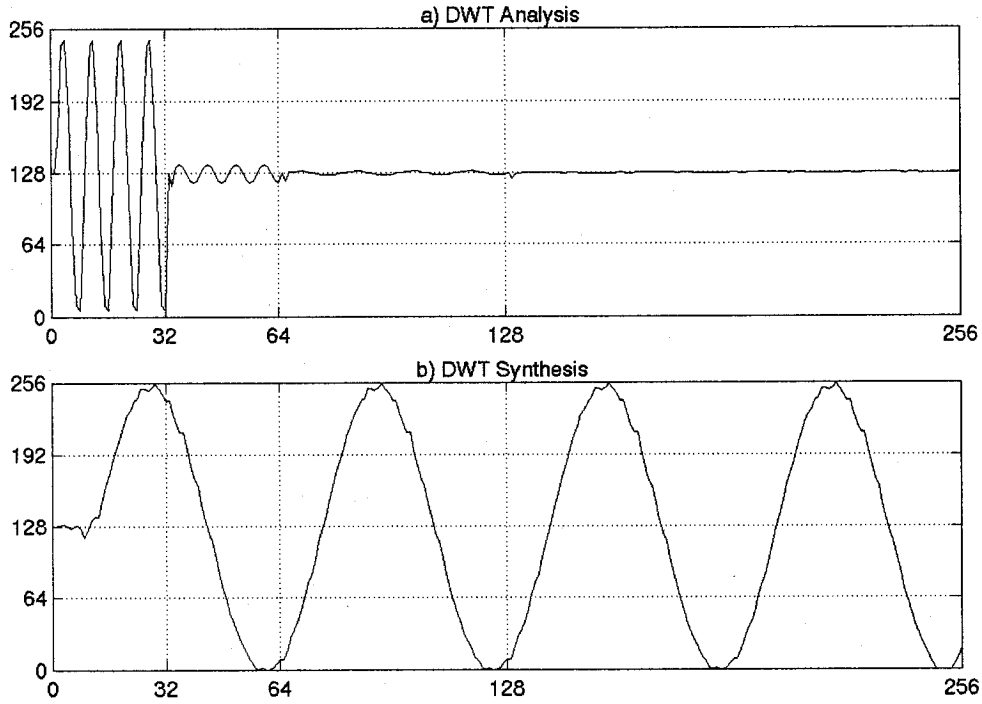

Figure 23. DWT Simulation with 9-bit Coefficients

4.3 DWT Image Processing Results

Two new Verilog modules were created to process the image according to the algorithm described in the paragraph 3.7. Both modules were designed to accept image data files in RAW format having 8-bit pixel words grouped in a data file of (**M** x **N**) bytes. "**M**" represents the number of columns "**N**" represents the number of rows. Both **M** and **N** are an integer multiple of eight.

The first module "*t_DWT_imageA.v*" encodes the original image into its DWT representation and the second module "*t_DWT_imageS.v*" decodes the DWT representation back to the original image.

An image processing freeware IrfanView available from "http://www.irfanview.com/" has been used to convert back and forth the images from their compressed form into RAW data.

The image-processing source "Neamt.gif" from Figure 24 represents a Romanian monastery from the XV century, and has a resolution of 640 x 480 pixels. The converted image "Neamt.raw" having 640 x 480 = 307,200 bytes was processed by the analysis module into "NeamtA.raw" and the converted result is shown in Figure 25. The analysis result "NeamtA.raw" was processed by the synthesis module and the reconstructed image is shown in Figure 26.

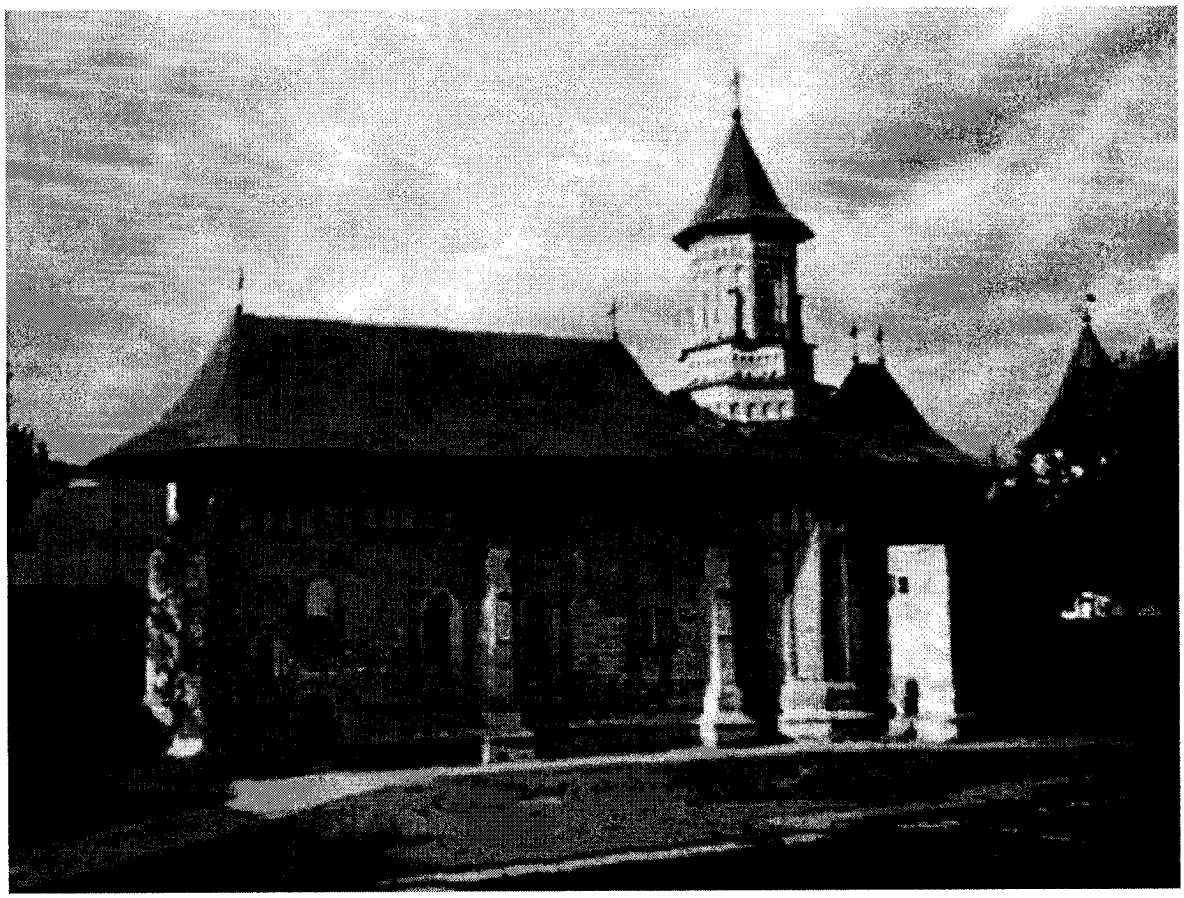

Figure 24. Original image "Neamt.gif"

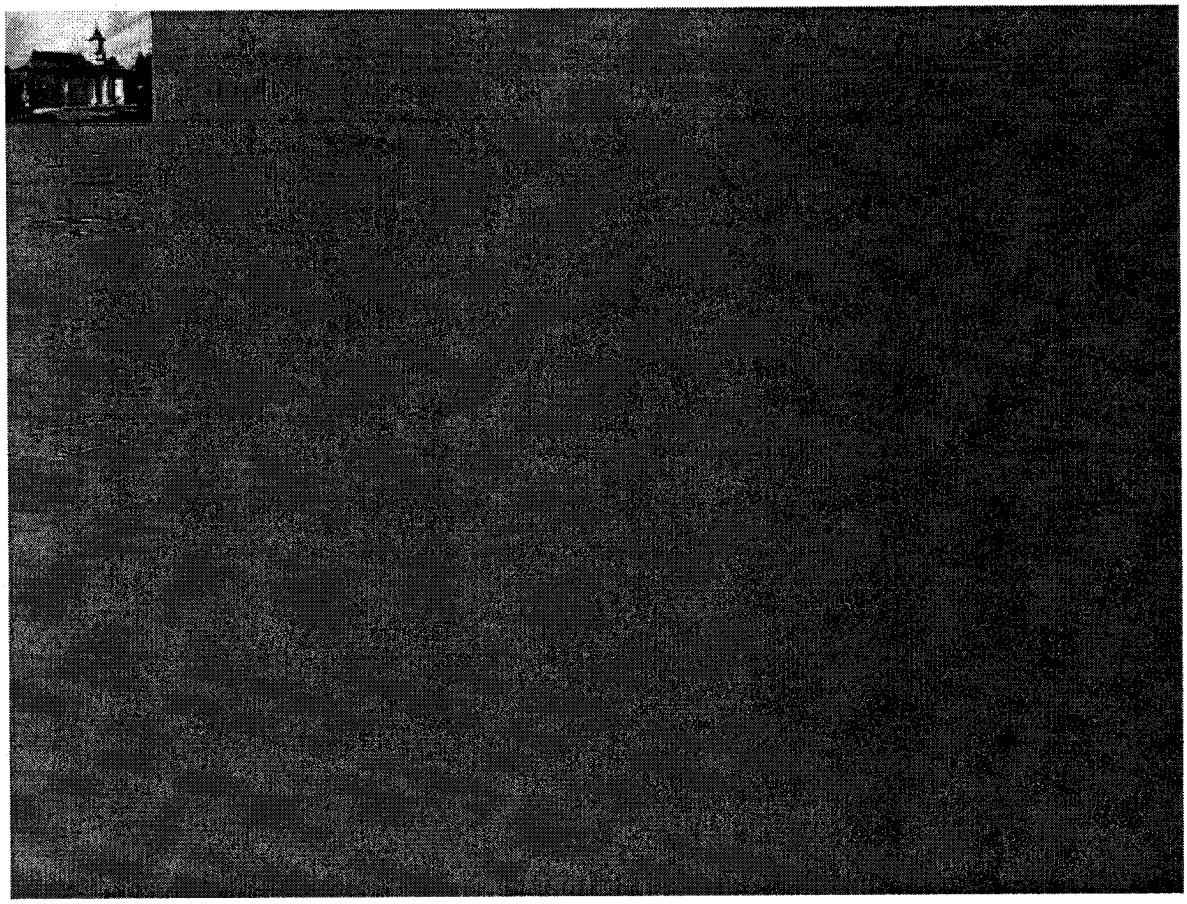

Figure 25. DWT Analysis image "Neamts.gif"

The reconstructed image in Figure 26 has a "shadow effect" for all the high contrast edges due to the four-tap FIR impulse response. Better results are achievable through using filters with higher number of taps (i.e. eight as described in [15]).

Figure 26. DWT Synthesis image "NeamtS.gif"

Chapter 5 Conclusion and Future Research

The objective of this work was to study, analyze and improve a DWT architecture that provides real-time coding for high-resolution video signal. A significant number of simulations were carried out to resolve problems related to fixed-point computation.

A systematic research program was elaborated to improve the architecture step by step. The first configuration had 16 bit Daubechies coefficients and 16 bit input samples. This configuration presented also in [16], was used to develop the *novel design method* described in paragraph 3.2. A number design flaws were identified at this level and the resulting implementation had a distorted waveform after reconstruction.

The process continued with the next implementation having 8-bit input samples and 24-bit coefficients. A *new Fast Processing Element* (see paragraph 3.4) was designed within this structure, and the resulting DWT synthesis had no distorted amplitude in most of cases and improved signal to noise ratio.

Further investigation revealed a *new solution* to the supra-unitary gain of the low-pass cascaded filter due to Daubechies coefficients. This solution was to *normalize and de-normalize* the filter coefficients with the same constant according to paragraph 3.6.

Since a 32-bit MAC structure was used from the first investigated architecture, a further optimized Fast Processing Element has been implemented with *8-bit* input samples combined with *25-bit* coefficients. This 8x25 MAC structure uses the full dynamic range

of the accumulator and accommodates supra-unitary coefficients used to de-normalize the DWT synthesis.

Three solutions were investigated to implement DWT normalization. The experimental results described in paragraph 4.2 have justified the theoretical judgment to choose the optimum scheme that uses the *sum of the low-pass filter coefficients* as normalization constant.

The last two simulations described in paragraph 4.2 revealed that an *8-bit by 9-bit MAC scheme* is providing the same quality transform as the ones that use larger number of bits for filter coefficient.

The modules included in the present architecture have been synthesized using *Zarlink Semiconductor 0.35μm CMOS technology*. The synthesized structure including 8-bit sample and 25-bit coefficient works at *75* MHz sample rate. The four-tap Fast Processing Element works at 150 MIPS, performing one multiply and one accumulate operation in the same cycle. This yields *2x150 = 300MIPS* for the two FPE included in both DWT analysis and DWT Synthesis module. The simulations for smaller bit size coefficients were performed by filling with zeros the least significant bits of the respective filter coefficients. An approximate three times smaller computation time is estimated for the 8-bit by 9-bit architecture that will process coefficients at 200MHz sample rate. The DWT Analysis module has 7281 gates and the DWT Synthesis module has 8214 gates (both estimations based on Synopsys area report). The *Matlab* tool was used to generate the input signal and to plot the results.

The final structure obtained was fully optimized to process video data within a row-by row and column by column 2D DWT architecture, as described in paragraph 3.7. Two image processing modules, one for *analysis* and another for *synthesis*, were successfully implemented to process the image contained in a video memory.

An area of future research starting from the present stage is *data compression*. The combined 2D DWT image coding together with adaptive or non-adaptive compression will provide a complete solution to high-speed the video compression requirement outlined in Chapter 1

In conclusion, an *optimal architecture for 8-bit Discrete Wavelet Transform image processing* has been successfully implemented within this work, combining three novel architectural solutions revealed in the present thesis. The proposed architecture is intended to be integrated in a *high-resolution video compression and decompression scheme*, within a future framework.

Chapter 6 References

[1] A.N. Akansu and M.S. Kadur, "Subband coding of video with adaptive vector quantization", *Proceedings ICASSP*, pages 2109-2112, Albuquerque, April 1990.

[2] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. "Image coding using wavelet transform", *IEEE Trans. Image Processing*,1(2):205-220, April 1992.

[3] M.R. Banham, J.C. Brailean, and A.K. Katsaggelos. "A wavelet transform sequence coder using nonstationary displacement estimation", *Proceedings of Visual Communications and Image Processing '92*, volume 1818, pages 210-221. SPIE, 1992.

[4] M. Barlaud, P. Sol_e, T. Gaidon, M. Antonini, and P. Mathieu, "Pyramidal lattice vector quantization for multiscale image coding", *IEEE Transactions on Image Processing*, 3:367-381, July 1994.

[5] P.J.Burt and E.H.Adelson, "The Laplacian pyramid as a compact image code", *IEEE Trans. Comm.*, COM-31: 532-540, April 1983.

[6] R.E.Crochiere, S.M.Weber, and J.K.L.Flanagan, "Digital coding of speech in sub-bands", *Bell Syst. Tech. Journal*, 55:1069-1086, Oct.1976.

[7] A.Croisier, D.Esteban, and C.Galand, "Perfect channel splitting by use of interpolation/decimation/tree decomposition techniques", *International Conf. On Inform. Sciences and Systems*, pages 443-446, Patras, Grece, August 1976.

[8] A. Cohen, I. Daubechies, and J.-C. Feauveau, "Biorthogonal bases of compactly supported wavelets" *Commun. on Pure and Appl. Math.*, 45:485-560, 1992.

[9] P.C. Cosman, K.L. Oehler, E.A. Riskin, and R.M. Gray, "Using vector quantization for image processing", *Proc. IEEE*, 81(9):1326-1341, Sep. 1993.

[10] I. Daubechies, "Orthonormal Bases of compactly supported wavelets", *Comm. on Pure Applied Math,* vol.41, pp.906-966, 1988.

[11] I. Daubechies, "Ten Lectures on Wavelets", *Society for Industrial and Applied Mathematics*, Philadelphia, 1992.

[12] I. Daubechies, W. Sweldens, "Factoring wavelet transforms into lifting steps", *Technical report, Bell Laboratories, Lucent Technologies*, September 1996.

[13] T. Denk, K.K. Parhi, and V. Cherkassky, "Combining neural networks and the wavelet transform for image compression", *Proceedings ICASSP*, volume 1, pages 637-640, Minneapolis, Minnesota, April 1993.

[14] J.C. Feauveau, P. Mathieu, M. Barlaud, and M. Antonini, "Recursive biorthogonal wavelet transform for image coding", *Proceedings ICASSP*, volume 4, pages 2649-2652, Toronto, Canada, May 1991.

[15] A. Grzesszczak. M.K. Mandal, S. Panchanathan, and T.H. Yeap, "VLSI Implementation of Discrete Wavelet Transform", *IEEE Trans. on VLSI Systems,* vol.4, no.4, pp.421-433, December 1996.

[16] E.Huluta, E.M. Petriu, S.R. Das, and A.H. Al-Dhaher, "Discrete Wavelet Transform Architecture Using Fast Processing Elements," *Proc. IMTC/2002, IEEE Instrum. Meas. Technol. Conf.*, pp. 1537-1542, Anchorage, AK, USA, May 2002.

[17] G. Kaiser, "A friendly guide to wavelets", Birkhauser, Boston, 1994.

[18] W. Li and Y. Zhang, "A study of vector transform coding of subband-decomposed images", *IEEE Trans. on Circuits and Systems for Video Technology*, 4(4): 383-391, Aug. 1994.

[19] S.G. Mallat, "Multifrequency channel decomposition of images and wavelet models", *IEEE Trans. on Acoustics, Speech, Signal Processing,* vol.37, pp.2091-2110, December 1989.

[20] S.G. Mallat, "A Wavelet Tour of Signal Processing", Academic Press, 1998.

[21] N. Moayeri, I. Daubechies, Q. Song, and H.S. Wang, "Wavelet transform image coding using trellis coded vector quantizers", *Proceedings ICASSP*, pages 405-408, 1992.

[22] I. Moccagatta and M. Kunt, "An image coding scheme based on perceptually classified VQ for high compression ratios", *Proceedings of the 1995 IEEE International Symposium on Acoustics, Speech, and Signal Processing*, pages 2487-2490, 1995.

[23] J.R. Ohm, "Advanced packet-video coding based on layered VQ and SBC techniques", *IEEE Trans. on Circuits and Systems for Video Technology*, 3(3):208-221, June 1993.

[24] K.K. Pahri and T. Nishitani, "VLSI Architectures for Discrete Wavelet Transforms", *IEEE Trans. on VLSI Systems,* vol.1, no.2, pp.191-202, June 1993.

[25] C.I. Podilchuk and N. Farvardin, "Perceptually based low bit rate video coding", *Proceedings ICASSP*, pages 2837-2840, Toronto,Canada, May 1991.

[26] Robi Polikar, "Multiresolution Analysis: the Discrete Wavelet Transform" "http://engineering.rowan.edu/~polikar/WAVELETS/WTpart4.html"

[27] O. Rioul and M. Vetterli, "Wavelets and signal processing", *IEEE Signal Processing Magazine,* 8(4): 14-38, October 1991.

[28] Matthias Schwab, "Transformation of discrete arrays by Daubechies wavelets", "http://sepww.standford.edu/public/docs/sep75/matt/paper_html/index.html"

[29] M. Vetterli, "Multi-dimensional sub-band coding: some theory and algorithms", *Signal Processing*, 6:97-112, April 1984.

[30] M. Vetterli and C. Herley, "Wavelets and filter banks: theory and design", *IEEE Trans. on Signal Processing,* vol.40, pp.2207-2232, September 1992.

[31] M. Vishwanath, "The recursive pyramid algorithm for Discrete Wavelet Transform", ", *IEEE Trans. on Signal Processing*, March 1994.

[32] M. Vishwanath and P. Chou, "An efficient algorithm for hierarchical compression of video", *Proceedings ICIP-94*, volume III, pages 275-279, Austin, Texas, November 1994. IEEE, IEEE Computer Society Press.

[33] Y.Q. Zhang and W. Li, "A study of non-separable subband filters for video coding", *Proceedings of Visual Communications and Image Processing* '92, volume 1818, pages 233-240. SPIE, 1992.

Annex A. Verilog modules

```
// Thesis:  DWT Architecture for Image Coding and Decoding
// Title:   Top module for DWT Analysis with normalized filter
// Module:  DWT_top_A
// Author:  E. Huluta

// Description: this module implements the Folded Three-level
//              DWT Analysis Architecture

module DWT_top_A(rstb, clkin, inpA, outA, ovf);

input  rstb, clkin;
input  [7:0] inpA;
output [7:0] outA;
output ovf;

/*
parameter g0 = 25'h1EF6F82;        // Daubechies coefficients for High-
pass filter
parameter g1 = 25'h1E34F41;
parameter g2 = 25'h06B12F7;
parameter g3 = 25'h1C22E45;
parameter h0 = 25'h03DD1BB;        // Daubechies coefficients for Low-
pass filer
parameter h1 = 25'h06B12F7;
parameter h2 = 25'h01CB0BF;
parameter h3 = 25'h1EF6F82;
*/

parameter g0 = 25'h1F44985;      // Daubechies coefficients/sum for High-
pass filter
parameter g1 = 25'h1EBB67B;
parameter g2 = 25'h04BB67B;
parameter g3 = 25'h1D44985;
parameter h0 = 25'h02BB67B;      // Daubechies coefficients/sum for Low-
pass filer
parameter h1 = 25'h04BB67B;
parameter h2 = 25'h0144985;
parameter h3 = 25'h1F44985;

/*
parameter g0 = 25'h1F4F501;      // Daubechies coefficients/1.5 for High-
pass filter
parameter g1 = 25'h1ECDF81;
parameter g2 = 25'h04761FA;
parameter g3 = 25'h1D6C984;
parameter h0 = 25'h029367C;      // Daubechies coefficients/1.5 for Low-
pass filer
parameter h1 = 25'h04761FA;
parameter h2 = 25'h013207F;
parameter h3 = 25'h1F4F501;
*/

/*
```

```
parameter g0 = 25'h1F7B7C1;      // Daubechies coefficients/2 for High-
pass filter
parameter g1 = 25'h1F1A7A0;
parameter g2 = 25'h035897C;
parameter g3 = 25'h1E11723;
parameter h0 = 25'h01EE8DD;      // Daubechies coefficients/2 for Low-
pass filer
parameter h1 = 25'h035897C;
parameter h2 = 25'h00E5860;
parameter h3 = 25'h1F7B7C1; */

// Controller instantiation
////////////////////////////////////////////////////

wire  fclk, sclk, s2k, s4k, s4k1, s8k;
wire  [1:0] csel;

DWTcntl_A control(rstb, clkin, fclk, sclk, s2k, s4k, s4k1, s8k, csel);


// Datapath structure
////////////////////////////////////////////////////////

reg  [7:0] D1, D2, D3;
wire [7:0] H_out, G_out;
wire ovfH, ovfG;
wire [7:0] mux0, mux01, mux1, mux11, mux2, mux21, mux3, mux31;
reg  [7:0] MUX;
wire [7:0] InR5;
reg  [7:0] R1, R2, R3, R4, R5, R6, R7;

// input multiplexing structure
assign mux0  = (s2k)  ? inpA  : mux01;
assign mux01 = (s4k1) ? H_out : R1;
assign mux1  = (s2k)  ? D1    : mux11;
assign mux11 = (s4k1) ? R2    : R5;
assign mux2  = (s2k)  ? D2    : mux21;
assign mux21 = (s4k1) ? R4    : R6;
assign mux3  = (s2k)  ? D3    : mux31;
assign mux31 = (s4k1) ? R6    : R7;

always @(csel or mux0 or mux1 or mux2 or mux3)
   case (csel)
      2'h0: MUX <= mux0;
      2'h1: MUX <= mux1;
      2'h2: MUX <= mux2;
      2'h3: MUX <= mux3;
   endcase

// Processing Elements instantiation
FPE pe_G(rstb, fclk, sclk, csel, g0, g1, g2, g3, MUX, G_out, ovfG);
FPE pe_H(rstb, fclk, sclk, csel, h0, h1, h2, h3, MUX, H_out, ovfH);

// output multiplexing structure
assign InR5 = (s4k1|s8k) ? R7 : R4;
assign outA = (s8k)      ? R4 : G_out;
```

```
assign ovf  = ovfG | ovfH;   // overflow signal

// Register instantiation
always @(posedge sclk or negedge rstb)
   if (~rstb) begin
      D1 <= 0; D2 <= 0; D3 <= 0;
      R1 <= 0; R2 <= 0; R3 <= 0; R4 <= 0; R5 <= 0; R6 <= 0; R7 <= 0;
   end else begin
      D1<=inpA;  D2<= D1; D3<= D2;
      R1<=H_out; R2<= R1; R3<= R2; R4<=R3; R5<=InR5; R6<=R5; R7<=R6;
   end

endmodule


// Thesis:  DWT Architecture for Image Coding and Decoding
// Title:   Top module for DWT Synthesis with normalized filter
// Module:  DWT_top_S
// Author:  E. Huluta

// Description: this module implements the Folded Three-level
//              DWT Synthesis Architecture

module DWT_top_S(rstb, clkin, inpS, outS, ovf);

input  rstb, clkin;
input  [7:0] inpS;
output [7:0] outS;
output ovf;

/*
parameter g0 = 25'h1EF6F82;        // Daubechies coefficients for High-
pass filter
parameter g1 = 25'h1E34F41;
parameter g2 = 25'h06B12F7;
parameter g3 = 25'h1C22E45;
parameter h0 = 25'h03DD1BB;        // Daubechies coefficients for Low-
pass filer
parameter h1 = 25'h06B12F7;
parameter h2 = 25'h01CB0BF;
parameter h3 = 25'h1EF6F82;
*/

parameter g0 = 25'h1E8930A;       // Daubechies coefficients*sum for
High-pass filter
parameter g1 = 25'h1D76CF6;
parameter g2 = 25'h0976CF6;
parameter g3 = 25'h1A8930A;
parameter h0 = 25'h0576CF6;       // Daubechies coefficients*sum for Low-
pass filer
parameter h1 = 25'h0976CF6;
parameter h2 = 25'h028930A;
parameter h3 = 25'h1E8930A;

/*
parameter g0 = 25'h1E72744;       // Daubechies coefficients*1.5 for
High-pass filter
```

```
parameter g1 = 25'h1D4F6E2;
parameter g2 = 25'h0A09C74;
parameter g3 = 25'h1A34568;
parameter h0 = 25'h05CBA98;        // Daubechies coefficients*1.5 for Low-
pass filer
parameter h1 = 25'h0A09C74;
parameter h2 = 25'h02B091E;
parameter h3 = 25'h1E72744;
*/

// Controller instantiation
////////////////////////////////////////////////
wire  fclk, sclk, s2k, s4k3, s8k, s8k1, s8k2, s8k5, s8k6;
wire  [1:0] csel;
DWTcntl_S control(rstb, clkin, fclk, sclk, s2k, s4k3, s8k, s8k1, s8k2,
s8k5, s8k6, csel);

// Datapath structure
////////////////////////////////////////////////////////
reg  [7:0] D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12;
wire [7:0] O_odd, O_even, out2;
wire ovfO, ovfE;
wire [7:0] mux0, mux01, mux1, mux2, mux21, mux22, mux3, mux31, mux32;
reg  [7:0] MUX;
wire [7:0] InR1, InR2, InR4;
reg  [7:0] R1, R2, R3, R4, R5;

// input multiplexing structure
assign mux0  = (s2k)       ? mux01 : D10;
assign mux01 = (s8k)       ? D4    : D8;
assign mux1  = D12;
assign mux2  = (s8k|s4k3)  ? mux21 : mux22;
assign mux21 = (s8k)       ? inpS  : O_even;
assign mux22 = (s8k6)      ? R1    : R2;
assign mux3  = (s8k|s8k2)  ? mux31 : mux32;
assign mux31 = (s8k)       ? D8    : R5;
assign mux32 = (s8k1|s8k5) ? R3    : R4;

always @(csel or mux0 or mux1 or mux2 or mux3)
   case (csel)
      2'h0: MUX <= mux0;
      2'h1: MUX <= mux1;
      2'h2: MUX <= mux2;
      2'h3: MUX <= mux3;
   endcase

// Processing Elements instantiation with coefficient compensation
FPE PE_even(rstb, fclk, sclk, csel, g3, g1, h3, h1, MUX, O_even, ovfE);
FPE PE_odd (rstb, fclk, sclk, csel, g2, g0, h2, h0, MUX, O_odd , ovfO);

// output multiplexing structure
assign InR1 = (s8k5) ? R4     : O_odd;
assign InR2 = (s2k)  ? R1     : O_even;
assign InR4 = (s8k5) ? R5     : R3;
assign outS = (s2k)  ? O_even : R1;

assign ovf  = ovfE | ovfO;   // overflow signal
```

```
// Register instantiation
always @(posedge sclk or negedge rstb)
   if (~rstb) begin
      D1 <= 0; D2 <= 0; D3 <= 0; D4 <= 0; D5 <= 0; D6 <= 0;
      D7 <= 0; D8 <= 0; D9 <= 0; D10<= 0; D11<= 0; D12<= 0;
      R1 <= 0; R2 <= 0; R3 <= 0; R4 <= 0; R5 <= 0;
   end else begin
      D1 <=inpS; D2 <= D1; D3 <= D2; D4 <= D3; D5 <= D4; D6 <= D5;
      D7 <= D6;  D8 <= D7; D9 <= D8; D10<= D9; D11<=D10; D12<= D11;
      R1<= InR1; R2<=InR2; R3 <= R2; R4<=InR4; R5 <= R4;
   end

endmodule


// Thesis:  DWT Architecture for Image Coding and Decoding
// Title:   Controller for DWT Analysis module
// Module:  DWTcntl_A
// Author:  E. Huluta

// Description: this module implements the controller for
//              DWT Analysis, using modulo-k counters

module DWTcntl_A(rstb, clkin, fclk, sclk, s2k, s4k, s4k1, s8k, csel);

input rstb, clkin;

output fclk, sclk, s2k, s4k, s4k1, s8k;
output [1:0] csel;

reg [5:0] scount;
reg s4k, s4k1, s8k;

always @(posedge clkin)      // synchronous counter for sclk*8
   if (~rstb)
      scount <= 0;
   else
      scount <= scount + 1;

assign fclk = ~scount[0];   // filter clock 4x sample clock (sclk)
assign sclk = ~scount[2];
assign csel =  scount[2:1];   // coefficient select
assign s2k  = ~scount[3];

always @(scount)      // modulo-k select signals
   case (scount[5:3])
      3'h0: begin s4k<=1; s4k1<=0; s8k<=1; end
      3'h1: begin s4k<=0; s4k1<=1; s8k<=0; end
      3'h2: begin s4k<=0; s4k1<=0; s8k<=0; end
      3'h3: begin s4k<=0; s4k1<=0; s8k<=0; end
      3'h4: begin s4k<=1; s4k1<=0; s8k<=0; end
      3'h5: begin s4k<=0; s4k1<=1; s8k<=0; end
      3'h6: begin s4k<=0; s4k1<=0; s8k<=0; end
      3'h7: begin s4k<=0; s4k1<=0; s8k<=0; end
   endcase
```

```
endmodule


// Thesis:  DWT Architecture for Image Coding and Decoding
// Title:   Controller for DWT Synthesis architecture
// Module:  DWTcntl_S
// Author:  E. Huluta

// Description: this module implements the controller for
//              DWT Synthesis, using modulo-k counters

module DWTcntl_S(rstb, clkin, fclk, sclk, s2k, s4k3, s8k, s8k1, s8k2,
s8k5, s8k6, csel);

input rstb, clkin;

output fclk, sclk, s2k, s4k3, s8k, s8k1, s8k2, s8k5, s8k6;
output [1:0] csel;

reg [5:0] scount;
reg s4k3, s8k, s8k1, s8k2, s8k5, s8k6;

always @(posedge clkin)      // synchronous counter for sclk*8
   if (~rstb)
      scount <= 0;
   else
      scount <= scount + 1;

assign fclk = ~scount[0];   // filter clock 4x sample clock (sclk)
assign sclk = ~scount[2];
assign csel =  scount[2:1];   // coefficient select
assign s2k  = ~scount[3];

always @(scount)      // modulo-k select signals
   case (scount[5:3])
      3'h0: begin s8k<=1; s8k1<=0; s8k2<=0; s4k3<=0; s8k5<=0; s8k6<=0;
end
      3'h1: begin s8k<=0; s8k1<=1; s8k2<=0; s4k3<=0; s8k5<=0; s8k6<=0;
end
      3'h2: begin s8k<=0; s8k1<=0; s8k2<=1; s4k3<=0; s8k5<=0; s8k6<=0;
end
      3'h3: begin s8k<=0; s8k1<=0; s8k2<=0; s4k3<=1; s8k5<=0; s8k6<=0;
end
      3'h4: begin s8k<=0; s8k1<=0; s8k2<=0; s4k3<=0; s8k5<=0; s8k6<=0;
end
      3'h5: begin s8k<=0; s8k1<=0; s8k2<=0; s4k3<=0; s8k5<=1; s8k6<=0;
end
      3'h6: begin s8k<=0; s8k1<=0; s8k2<=0; s4k3<=0; s8k5<=0; s8k6<=1;
end
      3'h7: begin s8k<=0; s8k1<=0; s8k2<=0; s4k3<=1; s8k5<=0; s8k6<=0;
end
   endcase

endmodule


// Thesis:  DWT Architectures for Image Coding and Decoding
```

```
// Title:   Fast Processing Element for DWT 4-tap normalized filter
// Module:  FPE
// Author:  E. Huluta

// Description: this module implements a 4-tap FIR using
//              a 8x25-bit multiply & 32-bit accumulate structure

module FPE(rstb, fclk, sclk, csel, c0, c1, c2, c3, in8, Dout, ovf);

input rstb, fclk, sclk;
input [1:0]  csel;
input [24:0] c0, c1, c2, c3;   // Daubechies coefficients
input [7:0] in8;

output [7:0] Dout;
output ovf;

reg [7:0] Dout;
reg [24:0] cmux;
reg [31:0] accum;
wire [31:0] prod, muxAcc, sum;
wire co;

// coefficient mux
always @(csel or c0 or c1 or c2 or c3)
   case (csel)
      2'h0: cmux <= c0;
      2'h1: cmux <= c1;
      2'h2: cmux <= c2;
      2'h3: cmux <= c3;
   endcase

mult8x25 multiplier(.x(cmux),.y(in8),.prod(prod));

// accumulator mux
assign muxAcc = (csel == 0) ? 32'h0 : accum ;

cs_adder
adder(.cin(prod[31]),.a(prod),.b(muxAcc),.sum(sum),.cout(co),.ovf(ovf));

always @(posedge fclk)       // accumulator
   accum <= sum;

always @(posedge sclk or negedge rstb)   // output register
   if(~rstb)
      Dout <= 0;
   else
      case (sum[31:30]) // 31:30/30:29 coefficient and overflow
compensation
         2'b00: Dout <= sum[30:23];
         2'b01: Dout <= 8'h7F; // 3F/7F positive overflow
         2'b10: Dout <= 8'h80; // C0/80 negative overflow
         2'b11: Dout <= sum[30:23];
     endcase

endmodule
```

```
// Thesis:  DWT Architecture for Image Coding and Decoding
// Title:   32-bit Carry Select Adder
// Module:  cs_adder
// Author:  E. Huluta

// Description: this module implements a 32-bit Carry Select Adder
//              using dual 4-bit full-adder cells

module cs_adder (cin, a, b, sum, cout, ovf);

input  [31:0]   a, b;
input           cin;
output [31:0]   sum;
output          cout, ovf;
wire            ovfn, ovfp;
wire   [31:0]   sum0, clamp;

// scf = sum carry false, sct = sum carry true
wire [31:0] scf1, sct1, scf2, sct2, scf3, sct3;

// cf = carry false, ct = carry true
wire cf11, cf12, ct12, cf13, ct13, cf14, ct14,
     cf15, ct15, cf16, ct16, cf17, ct17, cf18, ct18;
wire cf21, ct21, cf22, ct22, cf23, ct23, cf24, ct24;
wire cf31, ct31, cf32, ct32;

// groups of 4-bit full adders with carry select

assign {cf11,scf1[3:0]} = a[3:0]+b[3:0]+cin;

assign {cf12, scf1[7:4]} = a[7:4]+b[7:4];
assign {ct12, sct1[7:4]} = a[7:4]+b[7:4]+1;

assign {cf13, scf1[11:8]} = a[11:8]+b[11:8];
assign {ct13, sct1[11:8]} = a[11:8]+b[11:8]+1;

assign {cf14, scf1[15:12]} = a[15:12]+b[15:12];
assign {ct14, sct1[15:12]} = a[15:12]+b[15:12]+1;

assign {cf15, scf1[19:16]} = a[19:16]+b[19:16];
assign {ct15, sct1[19:16]} = a[19:16]+b[19:16]+1;

assign {cf16, scf1[23:20]} = a[23:20]+b[23:20];
assign {ct16, sct1[23:20]} = a[23:20]+b[23:20]+1;

assign {cf17, scf1[27:24]} = a[27:24]+b[27:24];
assign {ct17, sct1[27:24]} = a[27:24]+b[27:24]+1;

assign {cf18, scf1[31:28]} = a[31:28]+b[31:28];
assign {ct18, sct1[31:28]} = a[31:28]+b[31:28]+1;

// 1st stage carry select

assign scf2[3:0] = scf1[3:0];

assign {cf21, scf2[7:4]} = (cf11) ? {ct12,sct1[7:4]} : {cf12, scf1[7:4]};
```

```
assign scf2[11:8] = scf1[11:8];
assign sct2[11:8] = sct1[11:8];

assign {cf22, scf2[15:12]} = (cf13) ? {ct14, sct1[15:12]} : {cf14,
scf1[15:12]};
assign {ct22, sct2[15:12]} = (ct13) ? {ct14, sct1[15:12]} : {cf14,
scf1[15:12]};

assign scf2[19:16] = scf1[19:16];
assign sct2[19:16] = sct1[19:16];

assign {cf23, scf2[23:20]} = (cf15) ? {ct16, sct1[23:20]} : {cf16,
scf1[23:20]};
assign {ct23, sct2[23:20]} = (ct15) ? {ct16, sct1[23:20]} : {cf16,
scf1[23:20]};

assign scf2[27:24]  = scf1[27:24];
assign sct2[27:24]  = sct1[27:24];

assign {cf24, scf2[31:28]} = (cf17) ? {ct18, sct1[31:28]} : {cf18,
scf1[31:28]};
assign {ct24, sct2[31:28]} = (ct17) ? {ct18, sct1[31:28]} : {cf18,
scf1[31:28]};

// 2nd stage carry select

assign scf3[7:0] = scf2[7:0];

assign {cf31, scf3[15:8]} = (cf21) ? {ct22, sct2[15:8]} : {cf22,
scf2[15:8]};

assign scf3[23:16] = scf2[23:16];
assign sct3[23:16] = sct2[23:16];

assign {cf32, scf3[31:24]} = (cf23) ? {ct24, sct2[31:24]} : {cf24,
scf2[31:24]};
assign {ct32, sct3[31:24]} = (ct23) ? {ct24, sct2[31:24]} : {cf24,
scf2[31:24]};

// 3rd stage carry select

assign sum[15:0] = scf3[15:0] ;

assign {cout, sum[31:16]} = (cf31) ? {ct32, sct3[31:16]} : {cf32,
scf3[31:16]};

// Mark negative or positive overflow
assign ovfn = (a[31] & b[31] & ~sum[31]);
assign ovfp = (~a[31] & ~b[31] & sum[31]);
assign ovf = ovfn | ovfp;

endmodule


// Thesis:   DWT Architecture for Image Coding and Decoding
// Title:    8x25-bit Radix2 Multiplier
```

```
// Module:   mult8x25
// Author:   E. Huluta

// Description: This module multiplies an 8-bit number with a 25-bit
number
//              resulting a (31+1)-bit product. Parital 2's complement
//              result is further processed by Carry Select Adder.

module fad24(a, b, cin, s, cout);   // row of 16 full adder elements
input  [23:0] a,b;
output [23:0] s;
input  [23:0] cin;
output [23:0] cout;
  assign s    = ((a ^ b) ^ cin);
  assign cout =  (a & b) | ((a | b) & cin);
endmodule

module had24(a, b, s, cout);   // row of 16 half adder elements
input  [23:0] a,b;
output [23:0] s;
output [23:0] cout;
  assign s    = a ^ b ;
  assign cout = a & b ;
endmodule

module mult8x25(x, y, prod);

input  [24:0] x;
input  [7:0]  y;
wire   [23:0] xc;
wire   [6:0]  yc;
wire   [30:0] prod31, prod31c;
output [31:0] prod;
wire sign;      // the sign is two's complement carry for accumulator

// radix 2 multiplier wire structure
wire [23:0] s0, s1, s2, s3, s4, s5, s6;
wire [23:0] c0, c1, c2, c3, c4, c5, c6;
wire [23:0] p0, p1, p2, p3, p4, p5, p6;

// 2's complement muxes
assign xc = x[24] ? ~x[23:0] + 1 : x[23:0];
assign yc = y[7]  ? ~y[6:0]  + 1 : y[6:0];

// partial product terms
assign p0  =  {24{yc[0]}}  & xc;
assign p1  =  {24{yc[1]}}  & xc;
assign p2  =  {24{yc[2]}}  & xc;
assign p3  =  {24{yc[3]}}  & xc;
assign p4  =  {24{yc[4]}}  & xc;
assign p5  =  {24{yc[5]}}  & xc;
assign p6  =  {24{yc[6]}}  & xc;

// 1 half-adder and 13 full-adder rows connections
assign s0 = p0;   // first adder has two null inputs
had24  u1 (p1 ,{1'b0 , s0 [23:1]},  s1 , c1 );
fad24  u2 (p2 ,{1'b0 , s1 [23:1]},  c1 , s2 , c2 );
```

```
fad24  u3 (p3 ,{1'b0 , s2 [23:1]},  c2 , s3 , c3 );
fad24  u4 (p4 ,{1'b0 , s3 [23:1]},  c3 , s4 , c4 );
fad24  u5 (p5 ,{1'b0 , s4 [23:1]},  c4 , s5 , c5 );
fad24  u6 (p6 ,{1'b0 , s5 [23:1]},  c5 , s6 , c6 );

// unsigned product
assign prod31 = { (c6 + {1'b0, s6[23:1]}), s6[0], s5[0], s4[0], s3[0],
s2[0], s1[0], s0[0]};

// partial 2's complement result with carry to be added by cs_adder
// scaled to normal fractional number and fixed multiplication with 0
assign sign = (x[24] ^ y[7]) & |(y[7:0]);

// Multiplication with 8'h80 = -128 case
assign prod31c = (y==8'h80) ? { xc, 7'h00 } : prod31;
assign prod = { {sign}, (sign ? ~prod31c : prod31c) };

endmodule


// Thesis:  DWT Architecture for Image Coding and Decoding
// Title:   Testbench for DWT Analysis and Synthesis connected together
//          using normalized filter
// Module:  t_DWT_top
// Author:  E. Huluta

// Description: this module tests both DWT Analysis and Syntehsis

`timescale 1ns / 1ps
`define gsc200_V_LIB
file=/libs/cellib/DEL44/0p35gps/v1p4/libs/layout_libs/i1r3d0/gsc200/veril
og/gsc200.v
`uselib `gsc200_V_LIB

module t_DWT_top;

parameter buflen = 512;
parameter inputFile = "sinp8b512.dat";

parameter bp2 = buflen/2;
parameter bp4 = buflen/4;
parameter bp8 = buflen/8;
parameter fTime = (buflen+30)*40;  // provide enough time to process all
samples

reg [7:0] inMEM[0:buflen-1];   // sample memory
reg [7:0] anMEM[0:buflen-1];   // analysis memory
reg [7:0] snMEM[0:buflen-1];   // sythesis memory

integer i, fn0, fn1, fn2, fn3, fn4;
reg [20:0] lp8, hp8, hp4, hp2;  // memory pointers for DWT coefficients
reg [20:0] lps8, hps8, hps4, hps2;  // memory pointers for DWT
coefficients

reg  rstb, clkin;
reg  [20:0] scnt, syn_cnt;
reg  [7:0] inR, inp0, inps;
```

```
wire [7:0] inpa, inpsp, outa, outap, outs, outsp;
wire sclk, ovfa, ovfs;

always #2.5 clkin = ~clkin;

always @(posedge clkin)       // sample counter and modulo-8 counter
   if (~rstb)
      scnt <= 0;
   else
      scnt <= scnt + 1;

assign sclk = ~scnt[2];       // sample clock

// The Direct and Inverse DWT are connected together
DWT_top_A dwt_a(rstb, clkin, inpa, outa, ovfa);
DWT_top_S dwt_s(rstb, clkin, inpsp, outs, ovfs);

initial begin
   $readmemh(inputFile, inMEM);
   hp2 = bp2;  // initialize DWT analysis pointers
   hp4 = bp4;
   hp8 = bp8;
   lp8 = 0;
   hps2 = bp2;  // initialize DWT synthasis pointers
   hps4 = bp4;
   hps8 = bp8;
   lps8 = 0;
   syn_cnt =0;
   clkin = 0;
   inps = 8'h80;  // initialize synthesis input
   rstb = 1;
#4 rstb = 0;
#14 rstb = 1;
   inp0 = inMEM[0];  // Initial sample at DWT Analysis input
end


// Analysis structure
assign inpa = {~inp0[7], inp0[6:4], 4'b0};  // two's complement analysis
input
assign outap = { ~outa[7], outa[6:0] };  // two's complement analysis
output

always @(posedge sclk) begin
    inp0 <= inMEM[scnt[20:3]];
    if (scnt[20:3] <= buflen) begin  // end of transform detection
    #1  casex (scnt[5:3])
              3'b000: if (scnt[20:6] != 0) begin  // 3rd stage low-pass
filter output
                          anMEM[lp8] <= outap;
                      #1  lp8 <= lp8 + 1;  // increment pointer after
witing in memory
                      end
              3'b100: begin                // 3rd stage high-pass filter
output
                          anMEM[hp8] <= outap;
                      #1  hp8 <= hp8 + 1;
```

```
                    end
                3'bx10: begin                   // 2nd stage high-pass filter
output
                            anMEM[hp4] <= outap;
                        #1  hp4 <= hp4 + 1;
                        end
                3'bxx1: begin                   // 1st stage high-pass filter
output
                            anMEM[hp2] <= outap;
                        #1  hp2 <= hp2 + 1;
                        end
        endcase
    end
end


// Synthesis structure
assign inpsp = { ~inps[7], inps[6:0] };  // two's complement synthesis
input
assign outsp = { ~outs[7], outs[6:0] };  // two's complement synthesis
output

always @(posedge sclk) begin
    if (scnt[20:3] <= buflen)   // read all DWT analysis coefficients
    #2  casex (scnt[5:3])
                3'b000: begin   // 3rd stage low-pass filter coefficient
                        if (scnt[20:6] == 0)
                           inps <= 8'h80;  // two's complement of zero
                        else begin
                           inps <= anMEM[lps8];
                        #1 lps8 <= lps8 + 1;  // increment pointer after
witing in memory
                        end
                        end
                3'b100: begin  // 3rd stage high-pass filter output
                           inps <= anMEM[hps8];
                        #1 hps8 <= hps8 + 1;
                        end
                3'bx10: begin                   // 2nd stage high-pass filter
output
                           inps <= anMEM[hps4];
                        #1 hps4 <= hps4 + 1;
                        end
                3'bxx1: begin                   // 1st stage high-pass filter
output
                           inps <= anMEM[hps2];
                        #1 hps2 <= hps2 + 1;
                        end
         endcase
    else inps <= 8'h80;  // feed zero coefficients until synthesis is
finished
    if ((scnt[20:3] > 21) && (scnt[20:3] <= buflen + 21)) begin
    #3  snMEM[syn_cnt] <= outsp;     // Save synthesis coefficients
    #1  syn_cnt <= syn_cnt + 1;
    end
end
```

```
initial begin            // output files & waveforms
   $dumpfile("DWT_top_VCD.dump");
   $dumpvars();
#fTime  fn0 = $fopen("DWT_analysis");
   fn1 = $fopen("DWT_synthesis");
#5 for (i=0; i< buflen; i=i+1) begin // save DWT analysis/synthesis
results
       $fdisplay(fn0, anMEM[i]);
       $fdisplay(fn1, snMEM[i]);
   end
   $fclose(fn0);
   $fclose(fn1);
   $finish;
end

endmodule


// Thesis:  DWT Architecture for Image Coding and Decoding
// Title:   Testbench for DWT Analysis Controller
// Module:  t_DWTcntl_A
// Author:  E. Huluta

`timescale 1ns / 1ps
`define gsc200_V_LIB
file=/libs/cellib/DEL44/0p35gps/v1p4/libs/layout_libs/i1r3d0/gsc200/veril
og/gsc200.v
`uselib `gsc200_V_LIB

module t_DWTcntl_A;

reg rstb, clkin;

wire fclk, sclk, s2k,  s4k, s4k1, s8k;
wire [1:0]  csel;

DWTcntl_A DUT(rstb, clkin, fclk, sclk, s2k, s4k, s4k1, s8k, csel);

initial begin
      $dumpfile("DWTcntl_A_VCD");
      $dumpvars();
end

// 25MHz sample clock requires 25 x 8 = 200MHz clkin
always #2.5 clkin <= ~clkin;

initial begin
      clkin = 0;
      rstb = 1;
#7    rstb = 0;
#67   rstb = 1;
#500  $finish;
end

endmodule
```

```
// Thesis:  DWT Architecture for Image Coding and Decoding
// Title:   Testbench for DWT Synthesis Controller
// Module:  t_DWTcntl_S
// Author:  E. Huluta

`timescale 1ns / 1ps
`define gsc200_V_LIB
file=/libs/cellib/DEL44/0p35gps/v1p4/libs/layout_libs/i1r3d0/gsc200/veril
og/gsc200.v
`uselib `gsc200_V_LIB

module t_DWTcntl_S;

reg rstb, clkin;

wire fclk, sclk, s2k, s8k, s8k1, s8k2, s8k3, s8k4, s8k5;
wire [1:0]  csel;

DWTcntl_S DUT(rstb, clkin, fclk, sclk, s2k, s8k, s8k1, s8k2, s8k3, s8k4,
s8k5, csel);

initial begin
      $dumpfile("DWTcntl_S_VCD");
      $dumpvars();
end

// 25MHz sample clock requires 25 x 8 = 200MHz clkin
always #2.5 clkin <= ~clkin;

initial begin
      clkin = 0;
      rstb = 1;
#7    rstb = 0;
#67   rstb = 1;
#500  $finish;
end

endmodule


// Thesis:   DWT Architecture for Image Coding and Decoding
// Title:    Testbench for Carry Select Adder
// Module:   t_cs_adder
// Author:   E. Huluta

`timescale 1ns / 1ps
`define gsc200_V_LIB
file=/libs/cellib/DEL44/0p35gps/v1p4/libs/layout_libs/i1r3d0/gsc200/veril
og/gsc200.v
`uselib `gsc200_V_LIB

module t_cs_adder;

reg [31:0] a;
reg [31:0] b;
reg cin;
wire cout, ovf;
```

```
wire [31:0] sum;
reg clk;

cs_adder DUT(.cin(cin),.a(a),.b(b),.sum(sum),.cout(cout),.ovf(ovf));

initial begin
   $dumpfile("cs_adder_VCD");
   $dumpvars();
end

// clocked test to highlight the timming @ 100MHz
always #5 clk = ~clk;

initial begin
   a = 0;
   b = 0;
   cin = 0;
   clk = 0;
end

integer i;

initial begin
   // start_value = 32'h7f_ff_ff_ff;
   @(posedge clk);
   for(i=0;i<32;i=i+1)
   begin
      a = i;
      b = 32'h7f_ff_ff_ff;
      @(posedge clk);
      $display("cin:%b a:%h b:%h sum:%h cout:%b ovf:%b", cin, a, b, sum,
cout, ovf);
   end
   $finish;
end

endmodule

// Thesis:    DWT Architecture for Image Coding and Decoding
// Title:     Testbench for 8x25-bit Multiplier
// Module:    t_mult8x25
// Author:    E. Huluta

`timescale 1ns / 1ps
`define gsc200_V_LIB
file=/libs/cellib/DEL44/0p35gps/v1p4/libs/layout_libs/i1r3d0/gsc200/veril
og/gsc200.v
`uselib `gsc200_V_LIB

module  t_mult8x25;

reg   [24:0] x_in;
reg   [7:0]  y_in;
wire  [31:0] prod;

mult8x25 DUT(x_in, y_in, prod);
```

```
reg clk;
integer i, j, xi, yi, pi;

initial begin
      $dumpfile("mult8x25_VCD");
      $dumpvars();
end

initial begin
      clk  = 0;
      x_in = 0;
      y_in = 0;
end

// clocked test to highlight the timming @ 50MHz
always #5 clk <= ~clk;

initial begin
  @(posedge clk)
     for(i=0; i<10; i=i+1)
       for(j=0; j<10; j=j+1) begin
         x_in = 16777215 + i;
         xi =   16777215 + i;
         y_in = -128 + j;
         yi =   -128 + j;
         @(posedge clk)
           $display("%d %d %d %d", x_in, y_in, ~prod, xi*yi);
       end
$finish;
end

endmodule


// Thesis:  DWT Architecture for Image Coding and Decoding
// Title:   Testbench for Image DWT Analysis using normalized filter
// Module:  t_DWT_imageA
// Author:  E. Huluta

// Description: this module tests the DWT Image Analysis

`timescale 1ns / 1ps
`define gsc200_V_LIB
file=/libs/cellib/DEL44/0p35gps/v1p4/libs/layout_libs/i1r3d0/gsc200/veril
og/gsc200.v
`uselib `gsc200_V_LIB

module t_DWT_imageA;

parameter rowW = 640;
parameter columW = 480;
parameter maxBuf = 640 + 8;   // Additional space for null samples
processing
parameter imsize = rowW * columW;
parameter inputFile  = "Neamt.raw";
parameter outputFile = "NeamtA.raw";
// parameter inputFile  = "imodel.raw";
```

```
// parameter outputFile = "imodelA.raw";

parameter rTime = (40*(rowW+8) + 10)*columW + 3;  // provide enough time
to process all samples
parameter cTime = (40*(columW+8) + 10)*rowW + 3;  // provide enough time
to process all samples

reg [7:0] imMEM[0:imsize-1];   // image memory
reg [7:0] inBUF[0:maxBuf-1];   // row/column buffer
reg [7:0] anBUF[0:maxBuf-1];   // analysis buffer
reg [7:0] snBUF[0:maxBuf-1];   // sythesis buffer

integer i, frd, fn0, fn1;
reg [20:0] lpa8, hpa8, hpa4, hpa2;  // memory pointers for DWT
coefficients

reg  rstb, clkin;
reg  [20:0] scnt, syn_cnt;
reg  [7:0] inR, inp0, inps;
reg  [10:0] bcnt, buflen, bufp, columpW, rowpW;
wire [17:0] sample;
wire [7:0] inpa, inpsp, outa, outap, outs, outs2, outsp;
wire sclk, ovfa, ovfs, dumpBUF;

always #2.5 clkin = ~clkin;

always @(posedge clkin)       // sample counter and modulo-8 counter
   if (~rstb)
      scnt <= 0;
   else
      scnt <= scnt + 1;

assign sclk = ~scnt[2];       // sample clock

assign dumpBUF = (scnt[20:3] == buflen + 8);  // dump buffer + 8 for null
samples processing

always @(posedge dumpBUF) begin    // Dump buffer in memory
   if (~rstb)
       bcnt <= 0;
   else
       bcnt <= bcnt + 1;
   if (bcnt <= columW + rowW) begin
   #2  for (i=0; i<buflen; i=i+1)  // save analysis buffer in memory
            imMEM[(bcnt-columpW-1)*bufp + i*rowpW] <= anBUF[i];
       inps = 8'h80;               // initialize synthesis input
   #1  rstb = 0;
       if (bcnt >= columW) begin   // start of column processing
           buflen <= columW;
           bufp <= 1;
           columpW <= columW;
           rowpW <= rowW;
       end
   #2  for (i=0; i<buflen+8; i=i+1)  // load new row/column
           if (i < buflen)
               inBUF[i] <= imMEM[(bcnt-columpW)*bufp + i*rowpW];
           else
```

```
                inBUF[i] <= 0;
    #7  rstb = 1;
        inp0 = inBUF[0];            // Initial sample at DWT Analysis
input
        hpa2 = buflen/2;            // initialize DWT analysis pointers
for rows
        hpa4 = buflen/4;
        hpa8 = buflen/8;
        lpa8 = 0;
    end
end

// Analysis structure
assign inpa =  { ~inp0[7], inp0[6:0] };  // two's complement analysis
input
assign outap = { ~outa[7], outa[6:0] };  // two's complement analysis
output
assign sample = scnt[20:3];

always @(posedge sclk) begin
    inp0 <= inBUF[scnt[20:3]];
    if ((sample > 3) || (sample <= buflen+8)) begin // end of buffer
detection
    #1  casex (scnt[5:3])
                3'b000: if (scnt[20:6] > 1) begin    // 3rd stage low-pass
filter output
                            anBUF[lpa8] <= outap;
                        #1  lpa8 <= lpa8 + 1;        // increment pointer
after witing in memory
                        end
                3'b100: begin                        // 3rd stage high-
pass filter output
                            anBUF[hpa8] <= outap;
                        #1  hpa8 <= hpa8 + 1;
                        end
                3'bx10: begin                        // 2nd stage high-
pass filter output
                            anBUF[hpa4] <= outap;
                        #1  hpa4 <= hpa4 + 1;
                        end
                3'bxx1: begin                        // 1st stage high-
pass filter output
                            anBUF[hpa2] <= outap;
                        #1  hpa2 <= hpa2 + 1;
                        end
        endcase
    end
end

DWT_top_A dwt_a(rstb, clkin, inpa, outa, ovfa);

initial begin
        fn0 = $fopen(inputFile,"rb");
        frd = $fread(imMEM, fn0);
        $fclose(fn0);
        buflen = rowW;    // intialize row processing
        bufp = rowW;
```

```
        columpW = 0;
        rowpW = 1;
#1      for (i=0; i<buflen; i=i+1) begin
             inBUF[i] <= imMEM[i];
        end
        syn_cnt =0;        // initialize clocks
        scnt = 0;
        bcnt = 0;          // initialize buffer counter
        clkin = 1;
        inps = 8'h80;      // initialize synthesis input
        rstb = 1;
#2      rstb = 0;
#9      rstb = 1;
        inp0 = inBUF[0];   // Initial sample at DWT Analysis input
        hpa2 = buflen/2;   // initialize DWT analysis pointers for rows
        hpa4 = buflen/4;
        hpa8 = buflen/8;
        lpa8 = 0;
end

initial begin                              // save processed image
        $vcdpluson(1, t_DWT_imageA);
#rTime  ;
#cTime  fn1 = $fopen(outputFile,"wb"); // output files & waveforms
        for (i=0; i< imsize; i=i+1)    // save DWT analysis/synthesis
results
            $fwrite(fn1, "%1c", imMEM[i]);
        $fclose(fn1);
        $finish;
end

endmodule


// Thesis:  DWT Architecture for Image Coding and Decoding
// Title:   Testbench for Image DWT Synthesis using normalized filter
// Module:  t_DWT_imageS
// Author:  E. Huluta

// Description: this module tests the DWT Image Syntehsis

`timescale 1ns / 1ps
`define gsc200_V_LIB
file=/libs/cellib/DEL44/0p35gps/v1p4/libs/layout_libs/i1r3d0/gsc200/veril
og/gsc200.v
`uselib `gsc200_V_LIB

module t_DWT_imageS;

parameter rowW = 640;
parameter columW = 480;
// parameter pdly = 21;            // Processing delay inside the
structure
parameter pdly = 15;            // Processing delay inside the structure
parameter maxBuf = 640 + pdly;      // Additional space for null samples
processing
parameter imsize = rowW * columW;
```

```
parameter inputFile  = "NeamtA.raw";
parameter outputFile = "NeamtS.raw";
// parameter inputFile  = "imodelA.raw";
// parameter outputFile = "imodelS.raw";

// Pointers for row and column buffers
parameter rTime = (40*(rowW+pdly) + 10)*columW + 3;  // provide enough
time to process all samples
parameter cTime = (40*(columW+pdly) + 10)*rowW + 3;  // provide enough
time to process all samples

reg [7:0] imMEM[0:imsize-1];    // image memory
reg [7:0] anBUF[0:maxBuf-1];    // analysis buffer
reg [7:0] snBUF[0:maxBuf-1];    // sythesis buffer

integer i, frd, fn0, fn1;
reg [20:0] lps8, hps8, hps4, hps2;  // memory pointers for DWT synthesis
coefficients

reg  rstb, clkin;
reg  [20:0] scnt, syn_cnt;
reg  [7:0] inR, inp0, inps;
reg  [10:0] bcnt, buflen, bufp, columpW, rowpW;
wire [17:0] sample;
wire [7:0] inpa, inpsp, outa, outap, outs, outs2, outsp;
wire sclk, ovfa, ovfs, dumpBUF;

always #2.5 clkin = ~clkin;

always @(posedge clkin)       // sample counter and modulo-8 counter
   if (~rstb)
      scnt <= 0;
   else
      scnt <= scnt + 1;

assign sclk = ~scnt[2];       // sample clock

assign dumpBUF = (scnt[20:3] == buflen+pdly);  // dump buffer + pdly for
null samples processing

always @(posedge dumpBUF) begin    // Dump buffer in memory
   if (~rstb)
       bcnt <= 0;
   else
       bcnt <= bcnt + 1;
   if (bcnt <= columW + rowW) begin
   #2  for (i=0; i<buflen; i=i+1)  // save synthesis buffer in memory
           imMEM[(bcnt-columpW-1)*bufp + i*rowpW] <= snBUF[i];
   #1  rstb = 0;
       if (bcnt >= columW) begin   // start of column processing
           buflen = columW;
           bufp = 1;
           columpW = columW;
           rowpW = rowW;
       end
   #2  for (i=0; i<buflen+pdly; i=i+1)  // load new row/column
           if (i < buflen)
```

```
                anBUF[i] <= imMEM[(bcnt-columpW)*bufp + i*rowpW];
            else
                anBUF[i] <= 0;
   #7   rstb = 1;
        inps = anBUF[0];            // ??? columns Initial sample at DWT
Analysis input
        hps2 = buflen/2;            // initialize DWT analysis pointers
for rows
        hps4 = buflen/4;
        hps8 = buflen/8;
        lps8 = 1;                   // first element recorded
        syn_cnt = 0;                // initialize synthesis buffer counter
   end
end

assign sample = scnt[20:3];                 // sample counter

// Synthesis structure
assign inpsp = { ~inps[7], inps[6:0] };  // two's complement synthesis
input
DWT_top_S dwt_s(rstb, clkin, inpsp, outs, ovfs);
assign outsp = { ~outs[7], outs[6:0] };  // two's complement synthesis
output

always @(posedge sclk) begin
    if (scnt[20:3] <= buflen) begin  // read all DWT analysis
coefficients
        casex (scnt[5:3])
                3'b000: begin   // 3rd stage low-pass filter coefficient
                           inps <= anBUF[lps8];
                        #1 lps8 <= lps8 + 1;  // increment pointer after
witing in memory
                        end
                3'b100: begin  // 3rd stage high-pass filter output
                //            inps <= anBUF[hps8];
                           inps <= 0;
                        #1 hps8 <= hps8 + 1;
                        end
                3'bx10: begin                // 2nd stage high-pass filter
output
                //            inps <= anBUF[hps4];
                           inps <= 0;
                        #1 hps4 <= hps4 + 1;
                        end
                3'bxx1: begin                // 1st stage high-pass filter
output
                //            inps <= anBUF[hps2];
                           inps <= 0;
                        #1 hps2 <= hps2 + 1;
                        end
        endcase
    end else
        inps <= 8'h80;  // feed zero coefficients until synthesis is
finished
    if ((scnt[20:3] > pdly) && (scnt[20:3] <= buflen + pdly)) begin
      #1  snBUF[syn_cnt] <= outsp;    // Save synthesis coefficients
      #1  syn_cnt <= syn_cnt + 1;
```

```
    end
end

initial begin
        fn0 = $fopen(inputFile,"rb");
        frd = $fread(imMEM, fn0);
        $fclose(fn0);
#1      for (i=0; i<buflen; i=i+1) begin // read first row
             anBUF[i] <= imMEM[i];
        end
        syn_cnt =0;         // initialize clocks
        scnt = 0;
        bcnt = 0;           // initialize buffer counter
        clkin = 1;
        buflen = rowW;      // initialize row processinG
        bufp = rowW;
        columpW = 0;
        rowpW = 1;
        inps = anBUF[0];      // initialize synthesis input
        rstb = 1;
#2      rstb = 0;
#9      rstb = 1;
        hps2 = buflen/2;  // initialize DWT synthesis pointers for rows
        hps4 = buflen/4;
        hps8 = buflen/8;
        lps8 = 1;           // first element just recorded
end

initial begin                              // save processed image
        $vcdpluson(3, t_DWT_imageS);
// #rTime  ;
#rTime  fn1 = $fopen(outputFile,"wb"); // output files & waveforms
        for (i=0; i< imsize; i=i+1)     // save DWT analysis/synthesis
results
            $fwrite(fn1, "%1c", imMEM[i]);
        $fclose(fn1);
        $finish;
end

endmodule
```