A REAL-TIME, LOW-LATENCY, FPGA IMPLEMENTATION OF THE TWO DIMENSIONAL DISCRETE WAVELET TRANSFORM

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

OĞUZ BENDERLİ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

IN

THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

AUGUST 2003

Approval of the Graduate School of Natural and Applied Sciences

_____
Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____
Prof. Dr. Mübeccel Demirekler
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality, as a thesis for the degree of Master of Science.

_____
Assist. Prof. Dr. Yusuf Çağatay Tekmen
Supervisor

Examining Committee Members

Prof. Dr. Murat Aşkar (Chairman) _____

Assoc. Prof. Dr. Tayfun Akın _____

Assoc. Prof. Dr. Aydın Alatan _____

Assist. Prof. Dr. Yusuf Çağatay Tekmen _____

A. Neslin İsmailoğlu M.S. in EE _____

ABSTRACT

A REAL-TIME, LOW-LATENCY, FPGA IMPLEMENTATION OF THE TWO DIMENSIONAL DISCRETE WAVELET TRANSFORM

Benderli, Oğuz

M.S., Department of Electrical and Electronics Engineering

Supervisor: Assist. Prof. Dr. Yusuf Çağatay Tekmen

August 2003, 146 pages

This thesis presents an architecture and an FPGA implementation of the two dimensional discrete wavelet transformation (DWT) for applications where row-based raw image data is streamed in at high bandwidths and local buffering of the entire image is not feasible. The architecture is especially suited for multi-spectral imager systems, such as on board an imaging satellite, however can be used in any application where time to next image constraints require real-time processing of multiple images. The latency that is introduced as the images stream through the

DWT module and the amount of locally stored image data, is a function of the image and tile size. For an $n_1 \times n_2$ size image processed using $(n_1/k_1) \times (n_2/k_2)$ sized tiles the latency is equal to the time elapsed to accumulate a $(1/k_1)$ portion of one image. In addition, a $(2/k_1)$ portion of each image is buffered locally. The proposed hardware has been implemented on an FPGA and is part of a JPEG 2000 compression system designed as a payload for a low earth orbit (LEO) micro-satellite to be launched in September 2003. The architecture can achieve a throughput of up to 160Mbit/s. The latency introduced is 0.105 sec (6.25% of total transmission time) for tile sizes of 256×256. The local storage size required for the tiling operation is 2 MB. The internal storage requirement is 1536 pixels. Equivalent gate count for the design is 292,447.

Keywords: JPEG 2000, Wavelet Transform, FPGA, Multispectral Imaging, Image Processing

ÖZ

İKİ BOYUTLU AYRIK DALGACIK DÖNÜŞÜMÜNÜN, GERÇEK ZAMANLI VE DÜŞÜK GECİKMELİ OLARAK, FPGA UZERINDE GERÇEKLEŞTİRİLMESİ

Benderli, Oğuz

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Yrd. Doç. Dr. Yusuf Çağatay Tekmen

Ağustos 2003, 146 sayfa

Bu tezde, satır tabanlı ham görüntü verisinin yüksek bant genişliğinde duraksız iletildiği ve tüm veriyi yerel bellekte saklamanın mümkün olmadığı uygulamalara yönelik, iki boyutlu ayrık dalgacık dönüşümü (ADD) mimarisi ve FPGA gerçekleştirimi sunulmaktadır. Mimari, özellikle görüntüleme uydusu üzerinde bulunan çok bantlı görüntüleme sistemleri için uygun olup, birden fazla görüntünün gerçek zamanlı işlenmesini gerektiren ve görüntüler arası zamanlama kısıtının

olduğu uygulamalar için de kullanılabilir. Görüntüler ADD modülünden geçerken oluşan gecikme ve yerel olarak saklanması gereken görüntü verisi miktarı, görüntü ve parsel büyüklüğünün bir fonksiyonudur. $(n_1/k_1) \times (n_2/k_2)$ büyüklüğündeki parsellerle işlenen $n_1 \times n_2$ büyüklüğündeki bir görüntü için, gecikme zamanı, tüm resmin $(1/k_1)$ kadarlık bölümünün biriktirilmesi için geçen zaman kadardır. Ayrıca, tüm resmin $(2/k_1)$ kadarlık bölümü yerel olarak saklanmaktadır. Önerilen donanım, bir FPGA üzerinde gerçekleştirilmiştir ve Eylül 2003 tarihinde fırlatılıcak olan alçak yörüngeli bir mikro uydu için faydalı yük olarak tasarlanmış bir JPEG 2000 resim sıkıştırma sisteminin parçasıdır. Mimari 160 Mbits/s'e kadar veri işleyişi sağlayabilmektedir. 256×256'lık parsel boyutu için eklenen gecikme zamanı 0.105 saniyedir (Tüm iletim zamanının %6.25'i). Parselleme işlemi için gereken yerel bellek miktarı 2 MB'dir. İç bellek ihtiyacı ise 1536 pikseldir. Tasarımın eşdeğer geçit sayısı 292,447'dir.

Anahtar Kelimeler: JPEG 2000, Dalgacık Dönüşümü, FPGA, Çok Bantlı Görüntüleme, Görüntü İşleme

ACKNOWLEDGMENTS

This thesis would never have been written without the generous help and support that I received from numerous people along the way. I would now like to take this opportunity to express my sincerest thanks to these individuals.

First, I would like to express my appreciation to my supervisor Assist. Prof. Dr. Çağatay TEKMEN for his guidance and helpful comments in the development of this thesis. He has always been patient and supportive. Most of all, however, I would like to thank him for always having confidence in me and my abilities, even when I did not.

I would also like to express my acknowledgements and gratitude to Prof. Dr. Murat Aşkar, especially for his initiative role in the micro satellite and micro satellite payload projects of TÜBİTAK – BİLTEN. If it were not for him, this thesis would never have been started.

I would be remiss if I also did not thank my colleagues in TÜBİTAK – BİLTEN; first of all to my coordinator Neslin İsmailoğlu for her support and understanding throughout my thesis; to Soner Yeşil, Taner Kolçak, Ilgaz Korkmaz, Hacer Sunay and Refik Sever, who shared their bests in the GEZGIN project. I also wish to thank to TÜBİTAK-BİLTEN for facilities provided for the completion of this thesis.

I would also like to express my deep gratitude to my sincere friends, and my family for their continuous support, patience and encouragement.

to the memory of M. Emin Özkan

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

LIST OF ABBREVIATIONS

1-D	One Dimensional
2-D	Two Dimesnional
ASIC	Application Specific Integrated Circuit
BİLTEN	Bilgi Teknolojileri ve Elektronik Araştırma Enstitüsü
bpp	Bits per pixel
BRAM	Block Select Random Access Memory
CCD	Charge Coupled Device
ccs	Clock Cycles
CLB	Configurable Logic Block
CPU	Central Processing Unit
CY	Carry Symbol
DC	Direct Current (Convention for representing constant signals)
DCT	Discrete Cosine Transform
DFF	D Flip-flop
DLL	Delay Locked Loop
DSP	Digital Signal Processor

DWT	Discrete Wavelet Transform
EBCOT	Embedding Block Coding with Optimized Truncation
FF	Flip-Flop
FG	Function Generator
FPGA	Field Programmable Gate Array
GCLK	Global Clock Buffer
GEZGİN	Gerçek Zamanlı Görüntü İşleyen
HPI	Host Port Interface
Inf	Infinity
IC	Integrated Circuit
IO	Input/Output
IR	Infra-red
JPEG	Joint Photograph Experts Group
KLT	Karhunen Loeve Transform
LEO	Low Earth Orbit
LSB	Least Significant Bit
LUT	Look-up Table
MAC	Multiply-and-Accumulate
MRPA	Modified Recursive Pyramid Algorithm
MSB	Most Significan Bit
MSE	Mean Square Error
PE	Processing Element

PSNR	Peak Signal-to-Noise Ratio
QMF	Quadrature Mirror Filter
RAM	Random Access Memory
RCT	Reversible Color Transform
RGB	Red-Green-Blue
RPA	Recursive Pyramid Algorithm
SIMD	Single Instruction Multiple Data
SPIHT	Set Partitioning Hierarchical Trees
SSDR	Solid State Data Recorder
SSTL	Surrey Space Technologies Limited
TCQ	Trellis Coded Quantization
TÜBİTAK	Türkiye Bilimsel ve Teknik Araştırma Kurumu
VLSI	Very Large Scale Integrated Circuit
VQ	Vectorial Quantization

CHAPTER 1

INTRODUCTION

Digital imaging, whether it be professional or recreational, is a common reality today, allowing the capture of images using solid-state devices and image sensor devices instead of traditional film. The basic functioning of a digital camera is by means of recording the incident light through analog-to-digital conversion, thereby creating a digital representation of the image. Digital images have numerous advantages over traditional film images, such as ease of storage, access, transportation and manipulation.

For a digital image to be comparable in quality to an analog image generated through traditional film photography, a considerable amount of digital data should be stored. At 1200 dpi (dots per inch), a 5” by 4” image would translate into a 6000 pixel by 4800 pixel digital image, or 28.8 million pixels total. If each pixel is represented by 24 bits (8 bits for each spectral channel: Red, Green and Blue) this means storing roughly 9.1 Mbytes of digital data. Due to this large storage requirement, in digital imaging equipment, some compression algorithm is generally applied prior to storage.

Image compression algorithms comprise a sequence of treatment to image data such as transform, quantization, coding etc. *Two dimensional discrete wavelet transform*, 2-D DWT, is a powerful one of such transforms used in image compression. This work presents an architecture and an FPGA implementation of the two dimensional

discrete wavelet transform for applications where row-based image data is streamed in at high-bandwidths, and local buffering and random accessing the entire image is not feasible. The architecture is especially suited for , but not limited to, multi-spectral imager systems, such as on board an imaging satellite.

The proposed hardware has been implemented on an FPGA and is part of a JPEG 2000 compression system designed as a payload for a low earth orbit (LEO) micro-satellite, which will be launched in September 2003. The fundamental mission of the system is to process (compress) the output of digital imaging sensors in real-time, as the high bandwidth image data is output from the sensors, while storing only a small portion of the incoming image stream at any given time. This work includes the presentation and the report of the optimization of an ASIC co-processor which performs the required tasks before entropy coding in such a system.

This chapter is dedicated to providing introductions to several issues dealt with in this work and is organized as follows: In Section 1.1 a brief introduction to the concept of image compression is given followed by basics in wavelet transform in Section 1.2 and its application in image compression in Section 1.3. In Section 1.4 a method -which constitutes an important reference in this work- used to implement the wavelet transform using a lifting scheme is introduced. Section 1.5 is dedicated to the introduction of the new still image compression standard, JPEG 2000. In Section 1.6 introduces the application of the JPEG 2000 algorithm on-board the imaging satellite BILSAT-1.

In Chapter 2 hardware implementations of the 2-D DWT reported in related literature is presented. Chapter 3 presents in detail the proposed architecture for a 2-D DWT processor in a multi-spectral imaging application environment. Chapter 4 is dedicated to the implementation and simulation results. Chapter 5 summarizes the work done and results obtained, and possibilities of future work is mentioned.

1.1 Image Compression

An image is represented by a positive scalar function or –as a generalization- a multidimensional vector function of a plane. The value of this function at each point specifies the luminance or brightness of the color components of the picture at that point. Digital images are sampled versions of such functions, where the value of the function is specified only at discrete locations on the image plane, known as *pixels*. Luminance of each pixel is represented in a predefined precision of B bits. A typical value for B is eight which sufficiently accommodates the dynamic range of the human eye and is suitable for the commonly used computer memory structure since eight bit precision is suitable for existing computer memory structure (1 byte=8 bits).

The prevalent custom is that the samples (pixels) reside on a rectangular lattice of size $N_1 \times N_2$. The brightness at each pixel can be any number between 0 and $2^{B-1}-1$. The *raw representation* which is the simplest representation of an image is a list of matrix entries which give the brightness value of the corresponding pixel. The storage required for such a list is MN_1N_2B bits, where M is the number of color components.

In many imaging applications, exact reproduction of the image is not necessary. In this case, one can perturb the image slightly to obtain a shorter representation. If this perturbation is much smaller than the blurring and noise introduced in the formation of the image in the first place, there is no point in using the more accurate representation. Such a coding procedure, where perturbations reduce storage requirements, is known as *lossy* coding. The goal of lossy coding is to reproduce a given image with minimum distortion, given some constraint on the total number of bits in the coded representation.

The underlying reason that digital images can be compressed is the contained redundancy in representation of the images. The example of Nosratinia [1] can be given to illustrate this phenomenon: Suppose that we seek to efficiently store photographs of all natural scenes. In principle, we can enumerate all such pictures

and represent each image by its associated index. Assume we position hypothetical cameras at the vantage point of every atom in the universe (there are roughly 10^{80} of them), and with each of them take pictures in one trillion directions, with one trillion magnifications, exposure settings, and depths of field, and repeat this process one trillion times during each year in the past 10,000 years (once every 0.003 seconds). This will result in a total of 10^{144} images. But $10^{144} \approx 2^{479}$, which means that any image in this enormous ensemble can be represented with only 479 bits, or less than 60 bytes. This collection includes any image that a modern human eye has ever seen, including artwork, medical images, and so on, because we include pictures of everything in the universe from essentially every vantage point. And yet the collection can be conceptually represented with a small number of bits. If we assume that images are 512 × 512 and 8-bit, the remaining vast majority of the $2^{512\times512\times8} \approx 10^{600,000}$ possible images in the canonical representation are not of general interest because they contain little or no structure, and are noise-like.

The example illustrates the two main properties that image compression algorithms exploit: First, a very small fraction of the possible images that the representation provides are likely to be meaningful. If short code words for likely images and longer codewords for less likely images are used, a much shorter representation of the images can be achieved. This is the fundamental principle of operation of an *Entropy Coder*. Second, in our initial image gathering procedure we assign different representations for images which are visually indistinguishable from the other. Additional reductions in stored image size can be achieved by discretizing our database of images more coarsely. By mapping visually indistinguishable images to the same representation, we reduce the number of code words needed to encode images, at the price of a small amount of distortion.

Discretizing the database of images can be made by means of *quantizing.* Each pixel can be quantized separately, which is known as *scalar quantization,* or a group of pixels can be quantized together, which is called *vector quantization,* VQ. Since each pixel is quantized independent of the others, direct scalar quantization cannot capture the interdependency of the samples, and suffers of distortion at high

compression ratios. In principle, maximum compression that is theoretically possible can be achieved by VQ [1], however VQ reaches optimality only asymptotically as its dimensions increase. Furthermore, computational complexity and delay grow exponentially with the dimension of the VQ. limiting the practicality of VQ. Due to these and other difficulties, most practical image compression algorithms have turned to *transform coding.*

Transform coding consists of scalar quantization applied after a linear transform. This method captures much of the VQ gain, with only a fraction of the effort. Compression is performed in the transform domain. The main purpose of performing a transformation is to make the task of compression easier in the transform domain. Some of the well known transforms applied in transform coding are *the Karhunen-Loéve transform* (KLT)*, the discrete cosine transform* (DCT), and *sub-band transforms.*

The success of the transform coding depends on how well the basis functions of the transform represent the features of the signal. A good candidate transformation should be able to offer flexible image representation with decorrelation (to facilitate efficient entropy coding) and good energy compaction in the transform domain (so that fewer quantized coefficients are needed to be encoded and rest can be discarded with minimum distortion). At present, one of the most successful representations is the *wavelet transform,* application of which is a special case of sub-band transform [2]. In Section 1.2 and 1.3 a brief background of the wavelets and applications will be provided. For more background on wavelet theory and wavelet transform one can refer to [3-6].

State-of-the-art wavelet coders such as EZW [7], SPIHT [8], Trellis Coded quantization (TCQ) [9], EBCOT [10] are all derived from the transform coder paradigm. There are three basic components that underlie current wavelet coders: a decorrelating transform, a quantization procedure, and an entropy coding procedure. The next section provides a brief background to wavelets and explain why wavelets are useful for image compression.

1.2 Wavelet Basics

One of the most commonly used approaches for analyzing a signal $f(x)$ is to represent it as weighted sum of simple building blocks, called *basis* functions [11] :

$$f(x) = \sum_i c_i \Psi_i(x) \tag{1.1}$$

where $\Psi_i(x)$ are basis functions and the c_i are coefficients, or weights. $\Psi_i(x)$ are predefined fixed values, and therefore the signal information is contained by the coefficients. If we assume that $\Psi_i(x)$ are the translates of impulse function, this yields a representation in which coefficients only contain information about the time domain behavior of the signal. As an example for the other extreme, choosing sinusoids as the basis functions yields a Fourier representation that reveals information only about the signal's frequency domain behavior.

For the purpose of signal compression, neither of the representation is ideal. What we would like to have is a representation which contains information about both the time and frequency behavior of the signal. More specifically, a useful transformation should give the frequency content of the signal at a particular instant in time. However, resolution in time (Δx) and resolution in frequency ($\Delta\omega$) cannot both be made arbitrarily small at the same time because their product is lower bounded by the Heisenberg inequality [6].

$$\Delta x \Delta\omega \geq \frac{1}{2} \tag{1.2}$$

This inequality indicates that a trade off should be done between resolution in time an resolution in frequency. For example it is possible to obtain a good resolution in time if we are satisfied with the low resolution in frequency, and a good resolution in frequency if we are satisfied with the low resolution in time.

For efficient image compression, the aim is to use a transform in which transformed coefficients efficiently contain useful time-frequency information about a signal. By their very nature, low frequency events are spread out (or non-local) in time and

high frequency events are concentrated (or localized) in time. This means that if we split the signal's bandwidth in half, and repeat the halving operation on the low-pass portion of the bandwidth, we would have high-pass information analyzed by time-localized (but spread frequency) basis functions, and low-pass information would be analyzed by frequency-localized (but non-local in time) basis functions. Thus, an efficient representation of the signal would be possible.

Suppose that we have the impulse function as the basis function. The impulse function cannot provide information about the frequency behavior of a signal because its support –the interval over which it is non-zero– is infinitesimal. At the opposite extreme are the sinusoids, which cannot provide information about the time behavior of a signal because they have infinite support. Therefore, a compromise should be done between these two extremes: what should be chosen as basis functions, $\{\mathbf{\Psi}_i\}$, are those having finite support of different widths. The different support widths allow us to trade off time and frequency resolution in different ways; for example, we can analyze large regions of the signal and resolve low frequency details accurately by using wide basis functions, while we can use a short basis function to analyze a small region of the signal to resolve time details accurately.

Basis functions can be chosen as the scaled and translated version of the same prototype function Ψ, known as the *mother wavelet*. The scaling is an operation in which x is multiplied by a scale factor. If we choose the scale factor to be a power of 2, yielding $\mathbf{\Psi}(2^{\nu} x)$ where ν is some integer, we obtain a set of octave band-pass filters. Since Ψ has finite support, it will need to be translated along the time axis in order to cover an entire signal. This translation is accomplished by shifting Ψ in steps of size $2^{-\nu}k$, yielding;

$$\mathbf{\Psi}(2^{\nu} x - k), \qquad k \in \mathbf{Z} \tag{1.3}$$

With the new basis function, $\mathbf{\Psi}_{\nu k}(x)$, the *wavelet decomposition of the signal* is represented as :

$$f(x) = \sum_{\nu} \sum_{k} c_{\nu k} \Psi_{\nu k}(x), \tag{1.4}$$

where

$$\Psi_{\nu k}(x) = 2^{\frac{\nu}{2}} \Psi(2^{\nu} x - k) \tag{1.5}$$

In order to have an orthonormal set of basis, functions must be multiplied by $2^{\nu/2}$. The wavelet coefficients $c_{\nu k}$ are computed by the *wavelet transform,* which is just the inner product of the signal $f(x)$ with the basis functions $\Psi_{\nu k}(x)$:

$$c_{\nu k} = \langle f(x), \Psi_{\nu k}(x) \rangle \tag{1.6}$$

1.2.1 Application of Wavelet Transform in Image Compression

Wavelet-based image coding can be viewed as a form of a sub-band coding. The forward and inverse wavelet transforms can be implemented by a pair of *quadrature mirror filters* (QMFs). Each QMF pair consists of a low-pass filter, H, and a high-pass filter, G which split the input signal's bandwidth in half. The impulse responses of H and G are mirror images, and are related by :

$$g_n = (-1)^{1-n} h_{1-n} \tag{1.7}$$

The impulse response of the forward and inverse transform QMF's are related by :

$$g_n = \hat{g}_{-n} \tag{1.8a}$$

$$h_n = \hat{h}_{-n} \tag{1.8b}$$

Let $\tilde{g}_n$ and $\tilde{h}_n$ be the impulse responses of the forward transform and let g_n and h_n be the impulse responses of the inverse transform. Note that h_n is also the mother wavelet function of the orthogonal wavelet transform system.

Output of a filtering operation can be computed by convolving the filter coefficients $\widetilde{h}_k$ with the signal values:

$$\hat{s}_n = \sum_{k=0}^{L-1} \widetilde{h}_k s_{n-k} \tag{1.9}$$

where L is the number of coefficients, or in other words the *taps* of the filter. The one-dimensional forward wavelet transform of a signal s_n is performed by convolving s_n with both $\widetilde{h}_n$ and $\widetilde{g}_n$ and then down-sampling by 2:

$$s_n^{(l)} = \sum_{k=0}^{L-1} \widetilde{h}_k s_{2n-k} \tag{1.10a}$$

$$s_n^{(h)} = \sum_{k=0}^{L-1} \widetilde{g}_k s_{2n-k} \tag{1.10b}$$

Figure 1.1 shows the one dimensional forward and inverse wavelet transform. The low-pass output of the first stage is filtered for further levels[1]. Figure 1.2 illustrates the two dimensional separable forward wavelet transform for two dimensional signals. Note that throughout this discussion for a 2-D signal $f(x,y)$, x denotes the vertical axis and y denotes the horizontal axis.

[1] In literature recursive steps of wavelet transforming is referred to as either *octaves* or *levels.* In this work it is preferred to use the term *level* as in [19].

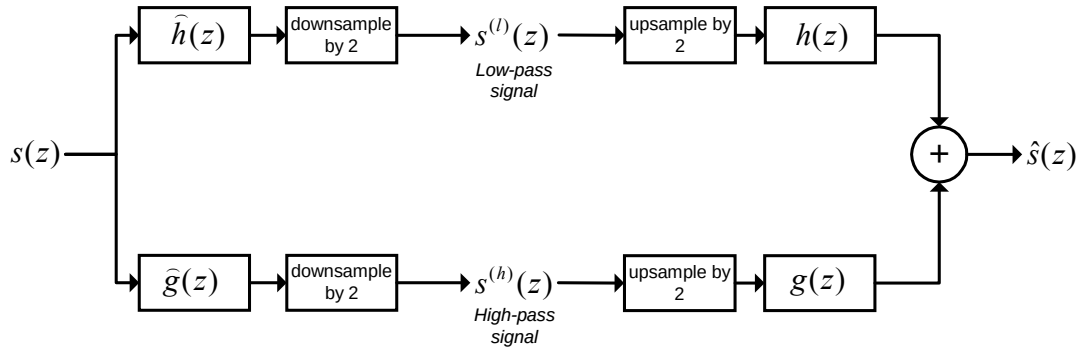


Figure 1.1 One dimensional forward and inverse wavelet transform. Each QMF pair consists of a low-pass filter, H, and a high-pass filter, G which split the input signal's bandwidth in half.

Owed to the separability of the filters, the transform can be performed in two steps each involving one dimensional filtering along different directions. The image $I(x,y)$ is first filtered along the y direction, resulting in a low-pass image and a high-pass image. Since the bandwidth of I along the y direction is split into two, we can safely down–sample each of the filtered images in the y direction by 2 without loss of information and obtain two images $L(x,y)$, and $H(x,y)$. The down-sampling or *decimation* is accomplished by dropping one sample in every two samples. Both $L(x,y)$ and $H(x,y)$ are then filtered along the x direcion, and once again we can down-sample the subimages by 2, this time along the x direction resulting four subimages (*sub-bands*) $LL(x,y)$, $LH(x,y)$, $HL(x,y)$, and $HH(x,y)$. As illustrated in Figure 1.2 the 2-D filtering decomposes an image into an *average signal* (LL) and three *detail signals* which are directionally sensitive: LH emphasizes the horizontal image features such as vertical edges and temporal changes along horizontal direction, HL emphasizes the vertical image features, and HH the diagonal features. The sensitivity of the detail signals is a result of the frequency ranges they contain.

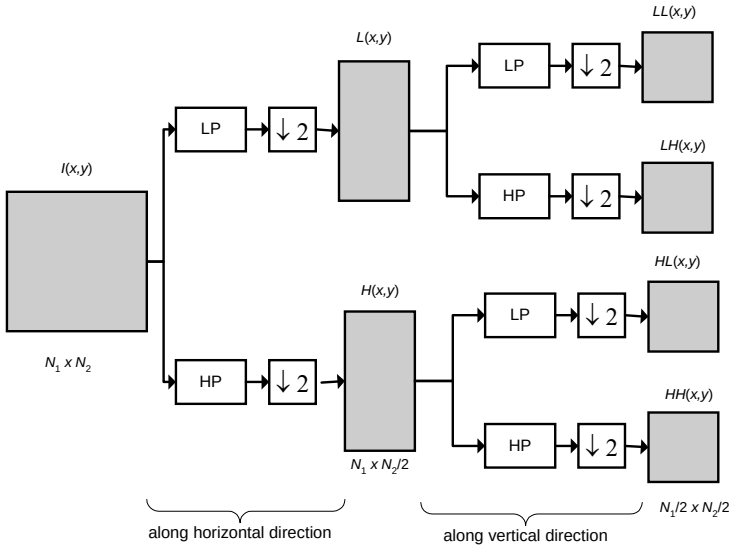


Figure 1.2 Two dimensional separable forward wavelet transform

In wavelet compression, the average signal is usually recursively transformed to higher levels as shown in Figure 1.3, the scheme was proposed by Mallat [12]. The wavelet decomposition is also called as *dyadic wavelet transform* or *Mallat tree decomposition.* The pass-band structure of the output signals is illustrated on Figure 1.4 for three levels of wavelet .

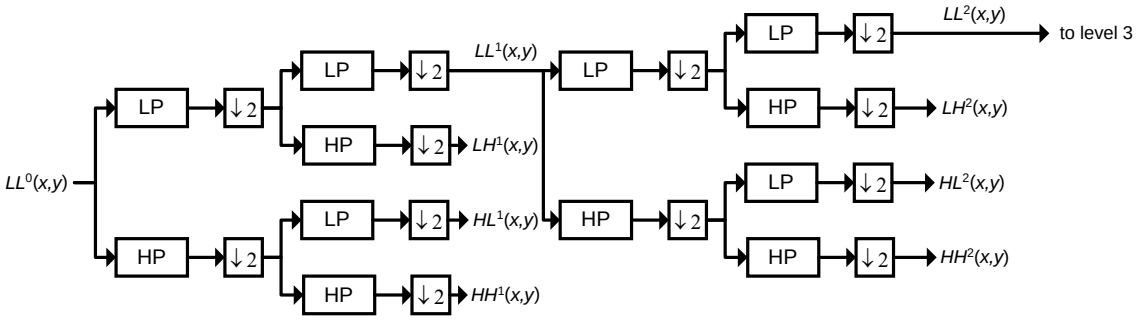


Figure 1.3 Dyadic (Mallat Tree) Wavelet Transform. In wavelet compression, the average signal is usually recursively transformed to higher levels.

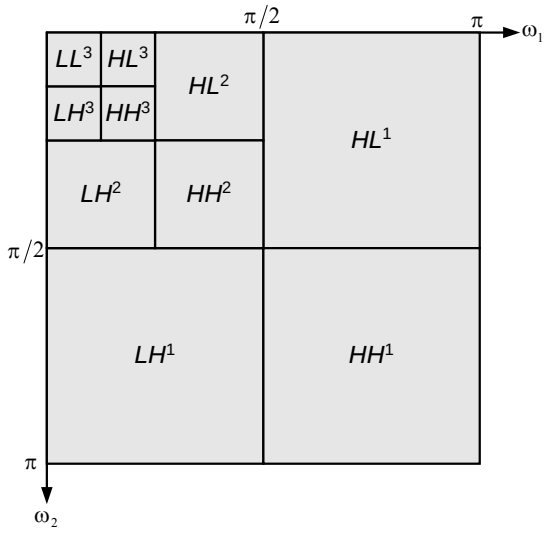


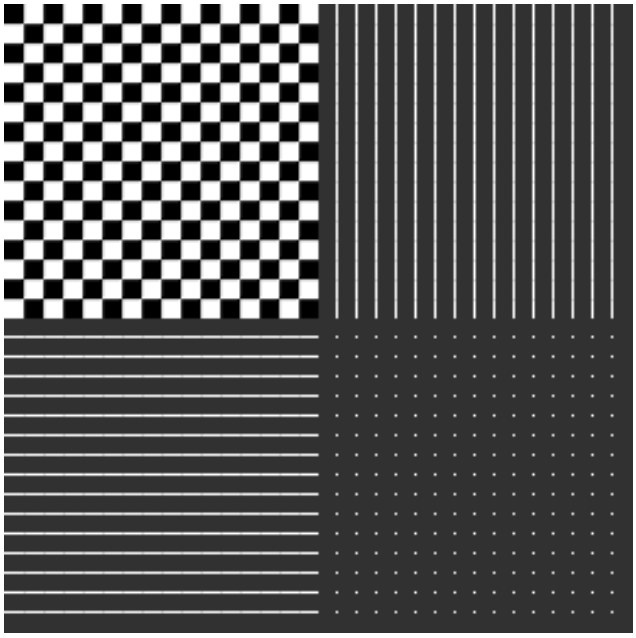
Figure 1.4 Pass-band structure of Sub-bands for three level wavelet transform

The number of transformations performed depends on several factors, including the amount of compression desired, the size of the original image, and the length of the QMF filters.

After the forward wavelet transform is completed, we have a matrix of coefficients which is equal in size to the original image containing the average signal and the detail signals of each scale. Up to this point we have accomplished no compression, moreover, each iteration of the forward wavelet transform causes the magnitude of the coefficients to grow, so the storage size for the image has actually been increased. Compression is achieved by quantizing and encoding the wavelet coefficients.

Figure 1.5 shows the wavelet transform of test images. 2.5a is a natural image and 2.5b is a checker board test image. Since the high-pass sub-bands contain samples centered around zero, absolute values of the samples with an offset is used for illustration purposes. Note that the vertical edges and temporal changes along horizontal direction are emphasized in HL sub-band whereas horizontal edges and temporal changes along vertical direction are emphasized in LH sub-band. The LL sub-band contains the coarse information of the image.

(a)



(b)

Figure 1.5 2-D Wavelet Transform outputs: (a) A natural image, (b) A checkerboard test pattern. The vertical edges and temporal changes along horizontal direction are emphasized in HL sub-band, whereas horizontal edges and temporal changes along vertical direction are emphasized in LH sub-band. The LL sub-band, the average signal, contains the coarse information of the image.

Reconstruction of the original image is by the 2-D inverse wavelet transform which is illustrated in Figure 1.6. The sub-bands are first up-sampled by 2 along the vertical axis (along *x*) and filtered along vertical axis with the corresponding inverse filters. LL and LH sub-bands are summed up to obtain sub-image L while HL and HH sub-bands are summed up to obtain sub-image H. Then the two sub-images are up-sampled along the horizontal axis (along *y*) and filtered with the corresponding inverse filters. Finally the outputs are summed up to obtain the reconstructed image, or the LL sub-band for the next iteration.

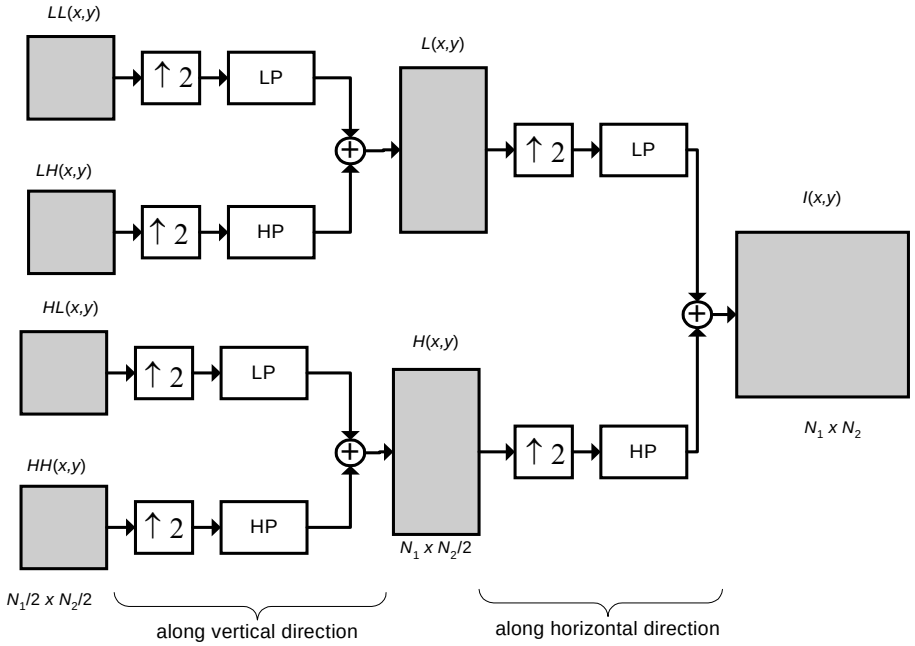


Figure 1.6 Two dimensional inverse wavelet transform

1.2.2 Factoring Wavelet Transforms into Lifting Steps

In this section a brief explanation of the lifting concept and how a QMF filter bank implementation of wavelet transform is factorized into lifting steps is given. More rigorous mathematical analysis of the subject can be found in [13-15].

Lifting factorization of a filter bank involves the *polyphase representation* of filter kernels. The polyphase representation of analysis filters $\widehat{h}$ and $\widehat{g}$ is given by :

$$\widehat{h}(z) = \widehat{h}_e(z^2) + z^{-1}\widehat{h}_o(z^2) \tag{1.11a}$$

$$\widehat{g}(z) = \widehat{g}_e(z^2) + z^{-1}\widehat{g}_o(z^2) \tag{1.11b}$$

where $\widehat{h}_e$ and $\widehat{g}_e$ contain the even coefficients, and $\widehat{h}_o$ and $\widehat{g}_o$ contain the odd coefficients. The wavelet transform of Figure 1.1 can be represented in polyphase form as illustrated in Figure 1.7. The input signal *s* is first split into even and odd parts, then the *polyphase matrix* is applied to the signals. In the inverse path, first polyphase matrix is applied and then the even and odd signals are joined properly. Polyphase matrix is given as:

$$\widehat{\mathbf{\Omega}}(z) = \begin{bmatrix} \widehat{h}_e(z) & \widehat{g}_e(z) \\ \widehat{h}_o(z) & \widehat{g}_o(z) \end{bmatrix} \tag{1.12}$$

and the outputs of the transform are expressed as :

$$\left[s^{(l)}(z) \;\; s^{(h)}(z)\right] = \left[s_e(z) \;\; s_o(z)\right] \cdot \widehat{\mathbf{\Omega}}(z) \tag{1.13}$$

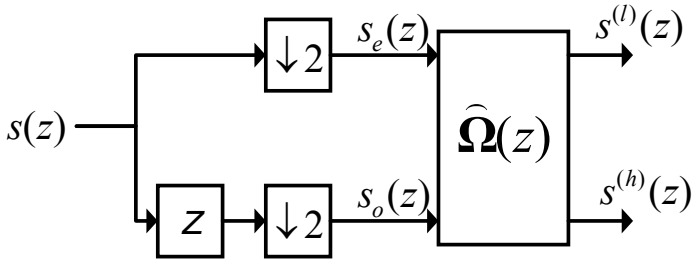


Figure 1.7 Polyphase representation of wavelet transform. The input signal *s* is first split into even and odd parts, then the *polyphase matrix* is applied to the signals.

The polyphase matrix can be factorized into several triangular matrices by *Euclidian algorithm* as follows:

$$\widehat{\mathbf{\Omega}}(z) = \left(\prod_{k=0}^{N-1} \begin{bmatrix} 1 & -\widehat{P}_k(z) \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ \widehat{U}_k(z) & 1 \end{bmatrix} \right) \cdot \begin{bmatrix} K_1 & 0 \\ 0 & K_2 \end{bmatrix} \tag{1.14}$$

where $\widehat{P}_k(z)$ and $\widehat{U}_k(z)$ are called *prediction* and *update* filters of the lifting steps implementation of wavelet transform, which is illustrated in Figure 1.8. First the low-pass samples (even terms) are filtered by prediction filters, $\widehat{P}_k(z)$, and are subtracted from the high-pass samples (odd terms) to obtain a "detail" signal. Then, the detail samples are filtered by the update filters, $\widehat{U}_k(z)$ and the low-pass samples are "updated" by adding the update filter outputs. This constitutes a single "lifting step" of the scheme. This lifting procedure is repeated as many times as the number of lifting steps *N*.

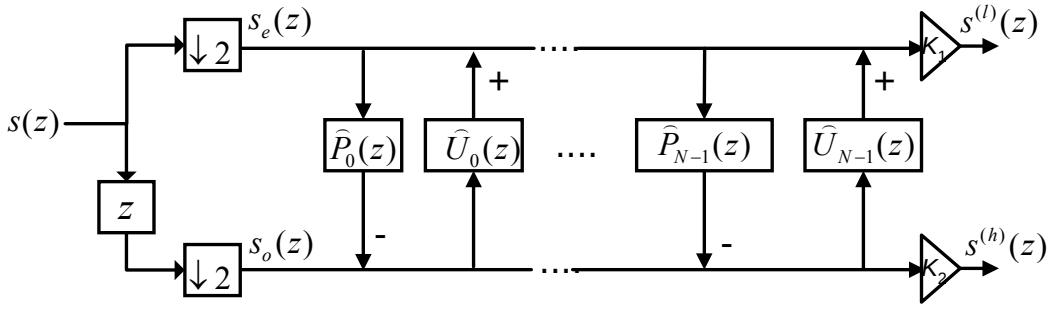


Figure 1.8 Lifting steps implementation of wavelet transform

The advantageous of lifting can be listed as follows:

1. It is easier to build non-linear wavelet transforms by using lifting. A typical example for non-linear transforms are the transforms that map integers to integers [15]. Such transforms are important for hardware implementations and for lossless image coding.

2. Every transform built with lifting is immediately invertible where the inverse transform has exactly the same computational complexity as the forward transform.

3. Lifting exposes the parallelism inherent in a wavelet transform. All operations within one lifting step can be done entirely parallel while the only sequential part is the order of the lifting operations.

4. Lifting involves poly-phase filtering which provides two channel input feeding, thus the clock cycle required to implement wavelet transform can be reduced.

1.3 A New Image Compression Standard : JPEG 2000

JPEG 2000 is an upcoming image compression standard published by the committee of JPEG, Joint Photographic Experts Group to serve the needs of current and future applications that uses still image coding. The committee's first published standard

JPEG is a simple and efficient discrete cosine transform (DCT) based lossy compression algorithm that uses Huffman Coding and is restricted to 8 bits/pixel. Though various extensions has appeared to JPEG to provide broader applicability and lossless compression, these extensions introduced only limited capability and faced with the intellectual copyright properties. Since 1996, various image compression algorithms were proposed and evaluated for the new image compression standard, and the one that was published at the end of 2000 by ISO (ISO I5444 | ITU-T Recommendation T.800) has been adopted as the new comprehensive still image compression standard, JPEG2000.

JPEG 2000 has many features, some of which are [16]

- Superior, Low bit-rate compression performance
- Progressive transmission by quality, resolution, component, or spatial locality
- Multiple resolution representation of still images
- Lossy and lossless compression
- Multispectral Image Support
- Random access to bit stream
- Pan and zoom (with decompression of only a subset of the compressed data)
- Compressed domain processing (eg. rotation and cropping)
- Region of interest coding
- More flexible file format
- Limited memory implementations
- Error Resilience

1.3.1 JPEG 2000 Coding Algorithm

In this section the JPEG 2000 algorithm is described. Figure 1.9 shows the block diagram of the JPEG 2000 coding algorithm. Comparative results are provided in [17-18].

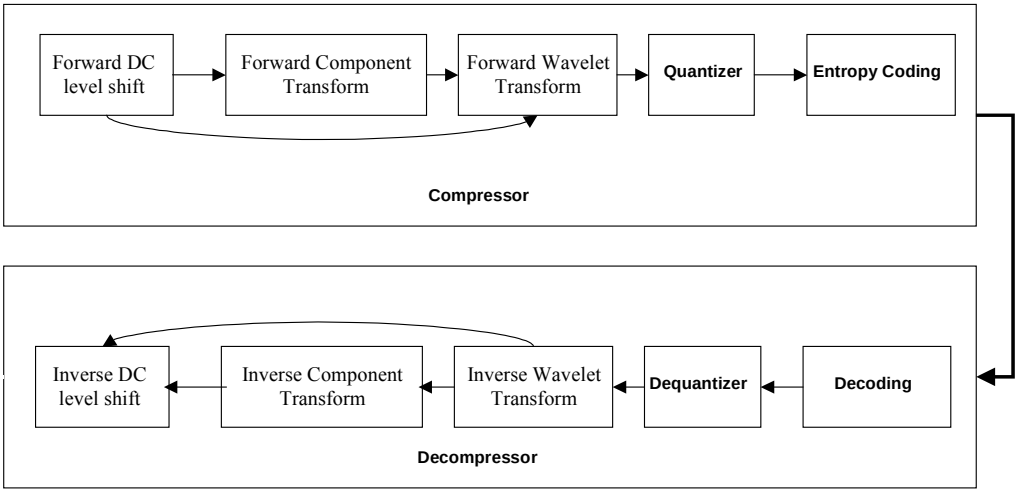


Figure 1.9 Block diagram of JPEG 2000 coding algorithm

The input image is first DC-level shifted, and then component transform is applied. For images having multiple color components, a point-wise decorrelating transform may be applied across the components. However this transform is optional. The standard Part I [19] defines 2 component transforms. These are : 1) the YCrCb transform commonly used in image compression systems and color format exchangers, and 2) the Reversible Component Transform (RCT) which provides similar decorrelation, but allows for lossless reconstruction of color components. Both color transforms are applied to first three components of the image data and the remaining components, if exist, are left unchanged. After the component transform, the image components are treated independently.

A color component can be processed in arbitrary sized non-overlapping rectangular blocks called *tiles* or the entire color component can be processed at a time (i.e. no tiles).

Given a tile, a *J*-level dyadic 2-D wavelet transform is applied. JPEG-2000 Part I offers two filtering methods which differ in filter kernels: Wavelet transform can be performed using either (9,7) filter, floating point wavelet [20], or (5,3) filter, integer wavelet [15]. For lossless compression (5,3) filter must be used. For a *J*-level transformation; from the lowest frequency sub-band (which is denoted in this work by S0), up to the *J*th resolution (which is denoted by S(*J*)), there are *J*+1 possible resolutions to reconstruct an image.

After wavelet transformation, uniform scalar quantization is applied to all sub-band coefficients. Uniform quantization involves a fixed dead-zone around zero. This corresponds to magnitude division and magnitude flooring. Further quantization can be applied during coding process by truncation of coefficients, thus rate control is achieved. For integer transform quantization step size is essentially –but not necessarily- 1, which means there is no pre-coding quantization, however rate control is achieved by truncation of coefficients as in floating-point transform.

After quantization each sub-band is subjected to *packet partition,* where each sub-band is divided into regular non-overlapping rectangles. After this step, *code-blocks* are obtained by dividing each packet partition location into regular non-overlapping rectangles. The code-blocks are the fundamental entities for the purpose of entropy coding.

Entropy coding is performed on each code-block independently. A context dependent, binary arithmetic coding is applied to bit planes of code-blocks. Algorithm employs the MQ-Coder which is defined in JBIG-2 standard [21] with some minor modifications.

1.4 GEZGİN: A JPEG 2000 Compression Sub-system On-board BILSAT-1

BİLSAT-1 [22] [68] is a 100kg class, low earth orbit (LEO), micro-satellite being constructed in accordance with a technology transfer agreement between TÜBİTAK-BİLTEN (Turkey) and SSTL (UK) and planned to be placed into a 650 km sun-synchronous orbit in Fall 2003. One of the missions of BİLSAT-1 is constructing a Digital Elevation Model of Turkey using both multi-spectral and panchromatic imagers. Due to limited down-link bandwidth and on-board storage capacity, employment of a real-time image compression scheme is highly advantageous for the mission.

Prof. Dr. Murat Aşkar has initiated resource and development projects which lead to the development of payloads for small satellites [68], one of which was planned to be an image processing subsystem while the other is a multi-spectral camera, ÇOBAN. GEZGİN [23] is a real-time image processing subsystem, developed for BILSAT-1. GEZGİN is one of the two R&D payloads hosted on BILSAT-1 in addition to the two primary imager payloads (a 4 band multi-spectral 26m GSD imager and a 12m GSD panchromatic imager). GEZGİN processes 4 images in parallel, each representing a spectral band (Red, Green, Blue and near Infra-Red) and captured by 2048 × 2048 CCD array type image sensors. Each image pixel is represented by 8-bits. The imaging mission of BILSAT-1 imposes a 5.5 seconds interval for real-time image processing between two consecutive multi-spectral images with 20% overlap in a 57 × 57km^2 swat. The image processing consists of streaming in the image data, compressing it with the JPEG2000 algorithm and forwarding the compressed multi-spectral image frames as a single stream to the Solid State Data Recorders (SSDR) of BILSAT-1 for storage and down-link transmission. Compression of image data in real-time is critical in micro-satellites in general, where the down-link and on-board storage capacity are limited. GEZGİN achieves concurrent compression of large multi-spectral images by employing a high degree of parallelism among image processing units.

The JPEG2000 compression on GEZGİN is distributed to dedicated Wavelet Transformation and Entropy Coding units. An SRAM based Field Programmable

Gate Array (FPGA) performs the computationally intensive tasks of image stream acquisition and wavelet transformation. A 32-bit floating-point Digital Signal Processor (DSP) implements the entropy coding (compression), formatting and streaming out of the compressed image data. The system allows for adjustment of the compression ratio to be applied to the images by means of run-time supplied quality measures. This results in great flexibility in the implementation of the JPEG2000 algorithm. Data flow into and out of GEZGİN is through dedicated high-speed links employing Low Voltage Differential Signalling (LVDS) at the physical layer. GEZGİN accommodates sufficient amount of on-board memory elements for temporary storage of the image data during acquisition and compression. The command/control interface of GEZGİN has an integrated Controller Area Network (CAN) bus. The configuration of the SRAM based FPGA together with the program code of the DSP can be uploaded in orbit through CAN bus, allowing for reconfiguration of the system.

CHAPTER 2

HARDWARE IMPLEMENTATIONS OF 2-D DISCRETE WAVELET TRANSFORMS, A LITERATURE REVIEW

The 2-D Discrete Wavelet Transform has a fundamental role in recently developed still and moving picture compression algorithms. However, because of its complexity in hardware implementations, a significant number of studies in the literature have been devoted to the design of architectures that effectively utilize available resources. Methods and algorithms have been proposed for the implementation of the 2-D DWT for the sake of simplifying the control circuitry, or architectures proposed for the implementations of such methods and algorithms. The publications in the literature are dedicated to proposing solutions for specific problems such as computation time, latency, memory requirements, routing complexity, inverse transform facilitation, utilization, etc.

This chapter is organized as follows: To provide a background, 1-D DWT architectures will be given in Section 2.1. Section 2.2, briefly discusses Mallat tree decomposition architectures, followed by a summary and a comparison of these architectures from an FPGA implementation perspective.

2.1 1-D DWT Architectures

Most 2-D DWT can be implemented with the use of one dimensional transform modules. Therefore, at this point, a brief background of one dimensional DWT architectures will be given. There have been a number of 1-D DWT architectures

studied so far [24-31], we will only discuss those relevant to hardware implementation of 2-D architectures.

Throughout this section N represents the length of the 1-D signal, L is the filter length, $x(n)$ is the input, $g(n)$ and $h(n)$ are the low-pass and high-pass filter outputs respectively. J denotes the maximum number of decomposition levels, and j is the current resolution level. Timing values are given in clock cycles (ccs) and storage sizes are given in terms of pixels.

2.1.1 Recursive Pyramid Algorithm Implementations

The recursive pyramid algorithm is a reformulation of the pyramid algorithm [12] introduced by Vishwanath [30]. It allows computation of the DWT in real-time, and provides an important storage size reduction. The algorithm uses storage of size $L(\log N - 1)$.

The output scheme is the linearized form of the pyramid structure. The algorithm is based on scheduling the outputs of any level j at the earliest instance that it can be scheduled. Instead of computing the jth level after the completion of j-1th level, the outputs from all levels are computed in an interleaved fashion. The outputs from the first level are computed once in every two received input sample. Therefore the higher levels can be interspersed between the first level. For an input of length N=8 and the decomposition level of J=3 the outputting schedule is as follows:

$$h_1(1), h_2(1), h_1(2), h_3(1), h_1(3), h_2(2), h_1(4), -, h_1(5), h_2(3), h_1(6), h_3(2), h_1(7), h_2(4), h_1(8), -$$

where $h_j(n)$ is the nth output of the jth level. (-) sign indicates that there is no scheduled output at that instance.

The pseudo code for the RPA is as follows :

```
begin {Recursive Pyramid}

   input : W[0,i]=x[i],    i:[1,N]  /* N is a power of 2 */
   low-pass filter  : h[m] m:[0,L-1]
   high-pass filter : g[m] m:[0,L-1]

   for (i=1 to N)                   /* Once for each output */
      rdwt(i,1)

end {Recursive Pyramid}


rdwt(i,j)


begin {rdwt}

   if (i is odd)
      k=(i+1)/2                     /* Compute output number k of level j
      sumL=0                        This is computed using the last L outputs
      sumH=0                        of level (j-1). */
      for (m=0 to (L-1))
         sumL=sumL+W[j-1,i-m]*h[m]
         sumH=sumH+W[j-1,i-m]*g[m]
      W[j,k]=sumL                   /* Low-pass output */
      W[j,k]=sumH                   /* High-pass output */

   else
      rdwt(i/2,j+1)                 /* Recursion to determine correct level */

end {rdwt}
```

Figure 2.1 Pseudo code for RPA

2.1.2 One Dimensional RPA Architectures

The first architecture for computing 1-D DWT was reported by Knowles [31]. Although this work was published before Vishwanath's RPA algorithm [30], this design can be classified as a RPA implementing architecture. Figure 2.2 shows the proposed DWT architecture.

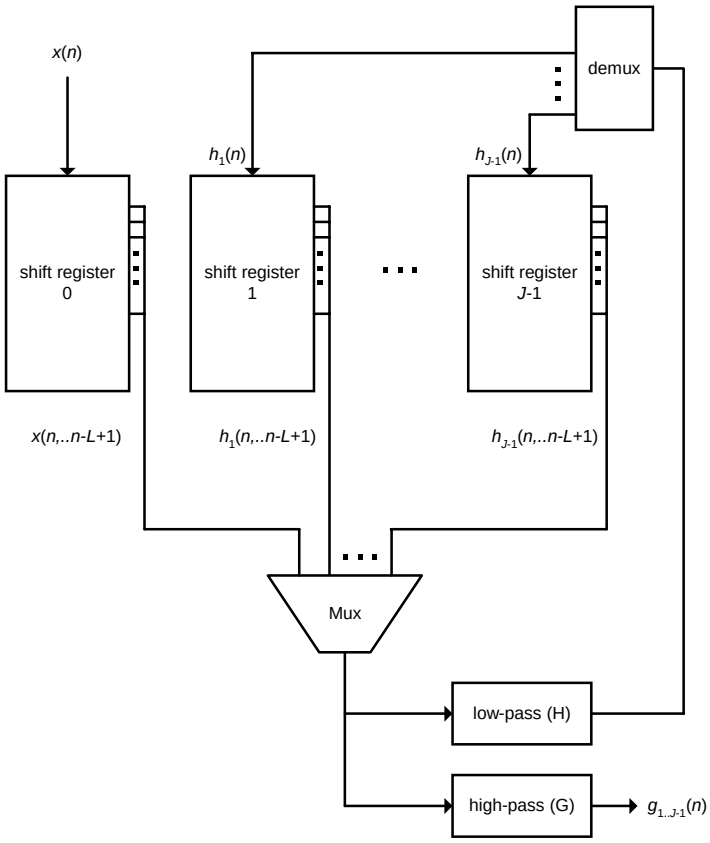


Figure 2.2 One dimensional RPA Architecture proposed by Knowles

The input $x(n)$ is loaded into an L depth shift-in parallel-out shift register. Where L is the maximum of the lengths of the low-pass and high-pass filters; $L = \max\{L_g, L_h\}$. Each intermediate result $h_j(n)$ obtained from the low-pass filter is also fed into a shift-in parallel-out shift register, while the high-pass filter outputs $g_j(n)$ are sent to output without being stored.

Since the architecture uses large multiplexors for routing intermediate results, it is not well suited for VLSI architectures. Several other architectures have been proposed in order to reduce the large routing introduced in DWT architectures.

2.1.2.1.Systolic/Semi-systolic Architectures

Viswanath proposed systolic and semi-systolic architectures which eliminate the wiring complexity in [32]. The architecture consists of filters handling low-pass and

high-pass filtering and a systolic routing network as shown in Figure 2.3. The routing network is a mesh of cells consisting of J-1 rows and L column, where J is the number of levels and L is the length of the filter.

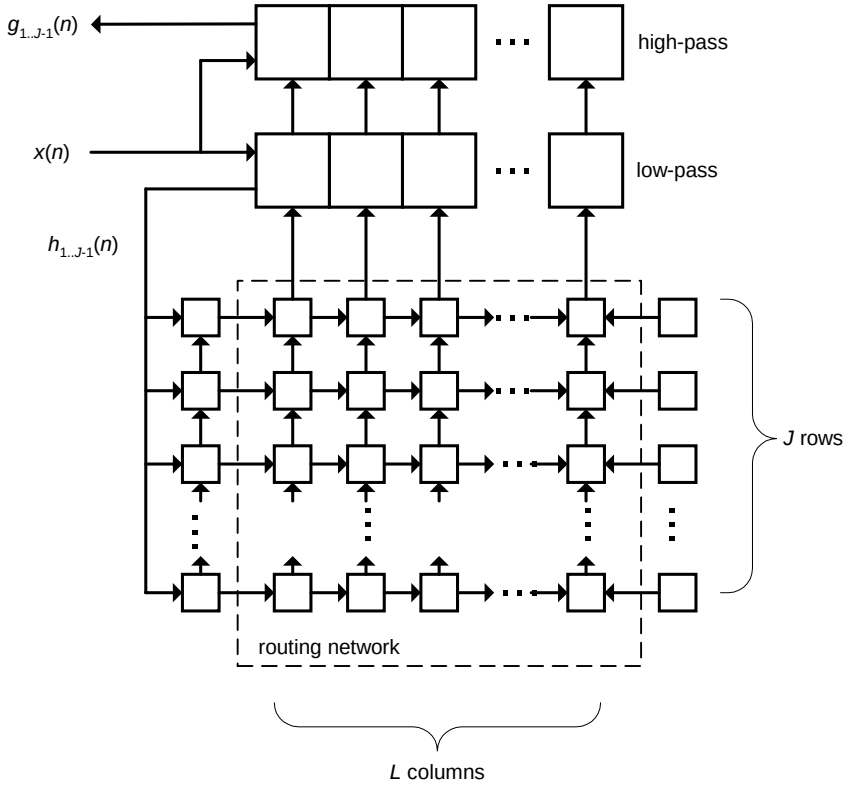


Figure 2.3 Systolic Architecture of Vishvanath

The architecture implements RPA as follows : The first level is computed conventionally in linear array and all the other levels are computed unconventionally. During the odd clock cycles each cell of the filters shifts in the input stream $x(n)$, while during the even cycles it takes the input from the proper level through the routing network. Thus the interspersion of higher levels between the first level is achieved.

In systolic structure the cells are capable of shifting data up and to the left. Several control signals such as shift-up shift-right, clock-up, clock-right are routed through

the network. The design of cells may be rather complex however the systolic network can be replaced with a semi-systolic one which provides global connections in vertical directions, eliminating the need for clock-up signals and extra control registers with the expense of wiring complexity.

2.1.2.2.Memory-Based Implementations

The routing network of systolic/semi-systolic architecture can be replaced with a RAM and address generators as shown in Figure 2.4.

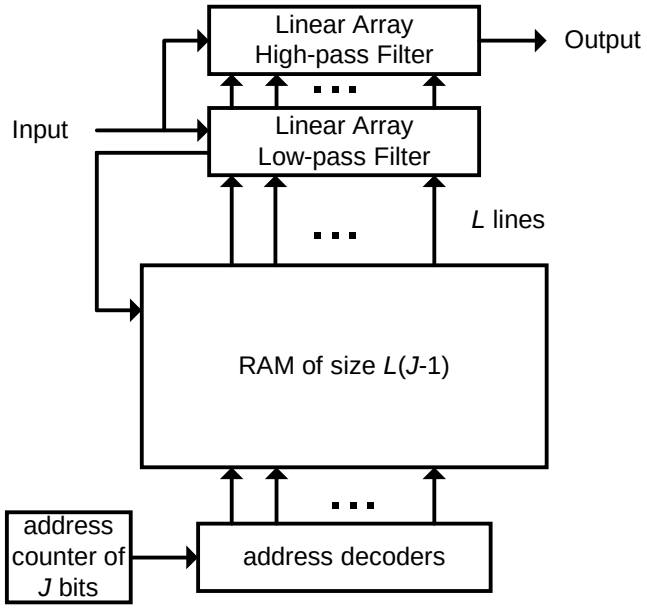


Figure 2.4 RAM based implementation of systolic architecture

2.1.2.3.Parallel Filtering

A similar structure to that of the systolic transformer in [32] is presented in [33], however with minor modifications. The x inputs are first fed into the storage instead of the linear array, and then loaded in parallel to the two parallel filters. This introduces a delay of L clock cycles.

The architecture is illustrated in Figure 2.5. The storage unit consists of J serial-in parallel-out shift registers each of length L. Each parallel filter consists of L multipliers and a tree of (L-1) adders. The parallel filter structure allows high sampling rates by adding pipe-lining stages, hence introduces computing latency to the filters. The latency introduced forces the use of a scheduling algorithm which takes into consideration this latency and is known as modified RPA (MRPA).

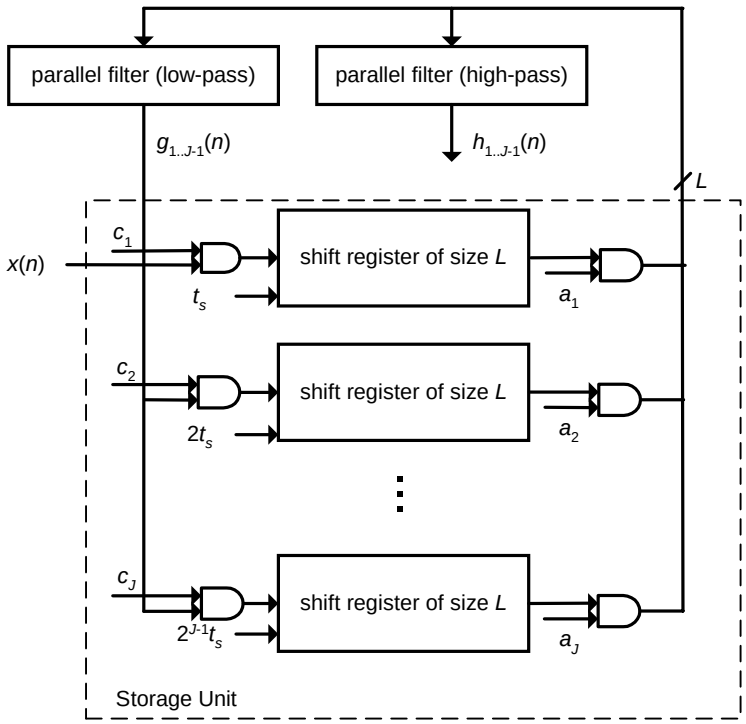


Figure 2.5 Parallel filter architecture

2.2 2-D Mallat Tree Decomposition Architectures

Mallat tree decomposition is the most popular of the 2-D wavelet transforms and is used in most image processing applications (see Figure 2.3). There are many Mallat tree decomposition architectures in the literature [27][33-45]. Some of these architectures will be discussed in two groups: Whole image storing and Memory-optimized. We will not consider off-chip data storing architectures such as [27][34-

35] since bottleneck of such implementations in high bandwidth applications is mainly the memory access not the architecture itself.

For the notation of Mallat tree, it is preferred to stick to the representation in [37]: High and low pass outputs after the j+1th stage computations along the row are denoted by $(LL)^jH$ and $(LL)^jL$ respectively. $(LL)^jHH$ and $(LL)^jHL$ denote the high-pass and low-pass outputs after computing along the columns of $(LL)^jH$ while $(LL)^jLH$ and $(LL)^jLL$ denote the high-pass and low-pass outputs after computing along the columns of $(LL)^jL$.

2.2.1 Whole Image Storing Structures

2.2.1.1.Direct Approach

Direct approach for the computation of 2-D wavelet transform is presented in [32]. The architecture implements Mallat tree decomposition with the repeated use of a single 1-D DWT module. Figure 2.6 shows the architecture of the 2-D DWT module. A memory of size equal to the input image, and an address generator which will handle the transposition of the intermediate coefficients is needed. To compute 1-D DWT of a row of length N, $2N$ clock cycles are needed hence the number of clock cycles needed to compute 1-D DWT of the whole image having N rows is $2N^2$. It is clear that $4N^2$ clock cycles are required in order to compute the 2-D DWT of the image. The first output is produced $2N^2$ cycles after the first input has arrived.

Despite the simplicity of the architecture, it may be an expensive task to store N^2 samples for large images. Moreover the latency of $2N^2$ cycles introduced before the first output is released may not be tolerable for many applications.

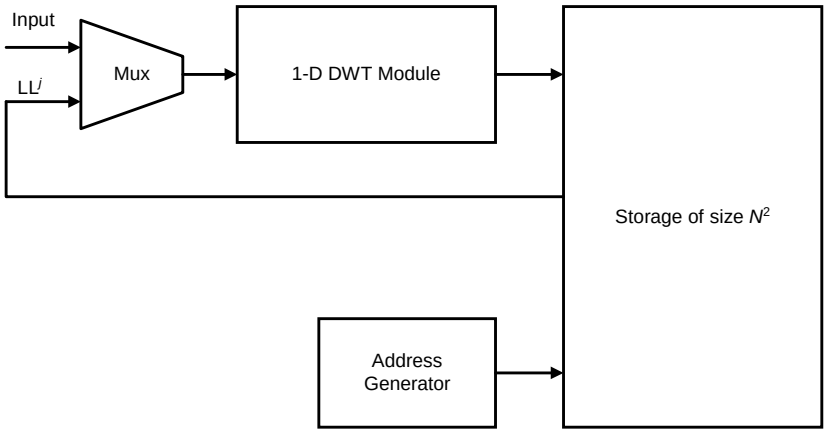


Figure 2.6 Direct Approach

2.2.1.2.Single Instruction Multiple Data (SIMD) Architectures

A SIMD architecture for the 2-D DWT is presented in [33], [37], where N x N image data is mapped onto an SIMD array of N x N processors. Each processor can be configured as 'active' or 'inactive' at each stage as follows: If a processor is set to 'inactive' it behaves like a simple wire which passes data through it with a negligible delay. If it is set to 'active', the processor does multiply-add operation and passes its data to the neighboring processor with a delay of one cycle. The interconnections of the 'active' processors for the first three stages where $N=8$ is shown in Figure 2.7. At each stage the high-pass and low-pass filter coefficients are broadcast to each processor; the coefficients and data are multiplied by 'active' processors and the partial results are updated. Then, data is shifted upwards during column operations and leftwards during row operations along the

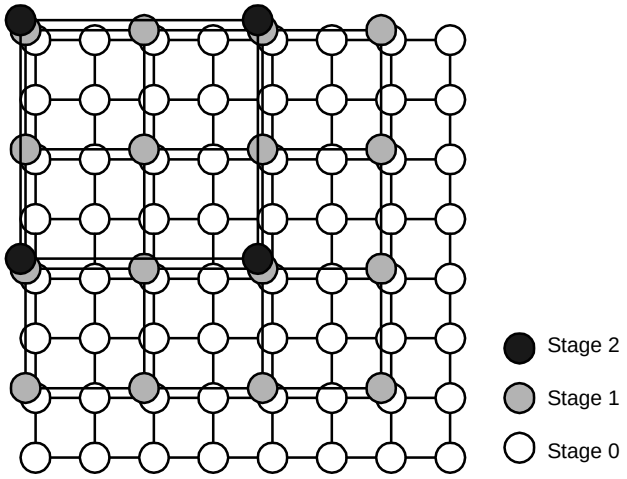


Figure 2.7 Interconnections of the 'active' processors for three stages where N=8

'active' processors. Another possible way of computation is to fix the data locations and shift the partial products. If each processor has only one multiplier at each stage row operations will take $2L$ cycles and column operations will take $2L$ cycles. This results a total processing time of $4JL$ cycles. Intermediate outputs of the processors are shown in Figure 2.8 for row and column operations. The memory requirement of the architecture is $2N^2$-$3N^2$.

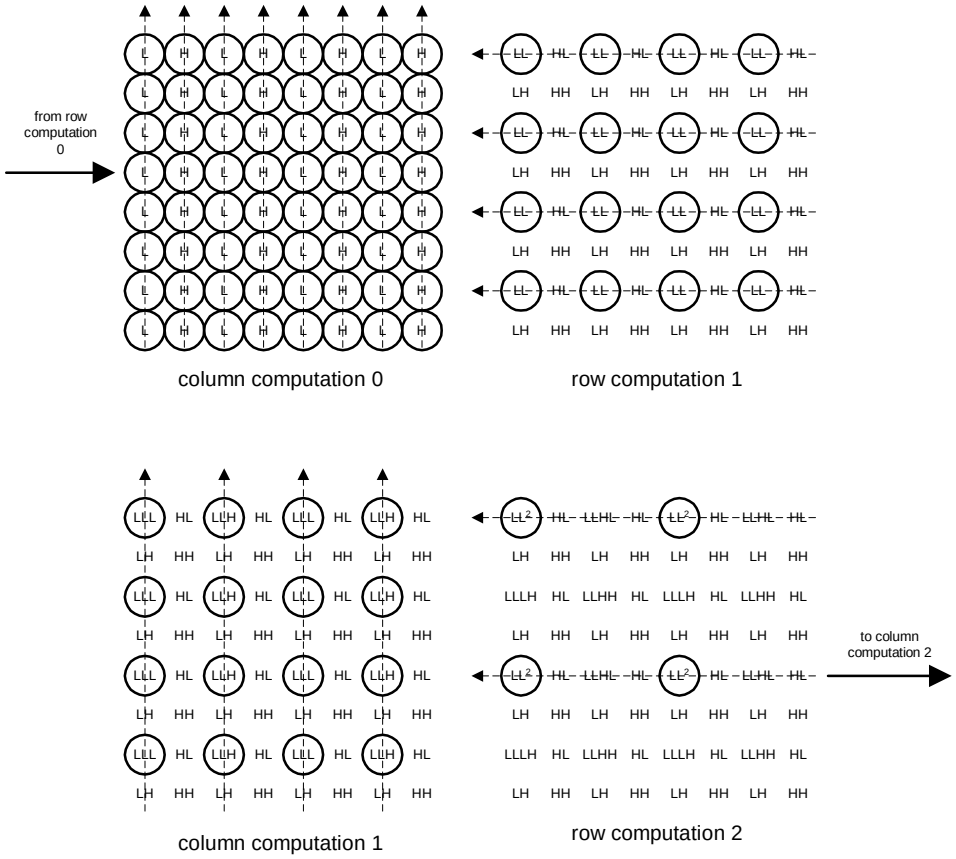


Figure 2.8 Intermediate outputs of the processors

2.2.2 Memory Optimized Architectures

2.2.2.1.Parallel Filter Algorithms

2-D discrete wavelet transforming using parallel filtering method explained above is presented in [33], [37] and [38]. Figure 2.9 shows the architecture for the 2-D DWT, where, filter 1 and 2 are the filters which operate on the rows and filter 3 and 4 are the filters which operate along the columns. the outputs of filter 1 (L and H) and the outputs of filter 2 ($(LL)^{j-1}L$ and $(LL)^{j-1}H$) are stored in storage 1 and read out by filters 3 and 4 in transposed form. Similarly, the output of filter 3, $(LL)^{j}$ is stored in storage 2 and read out in transposed form by filter 2.

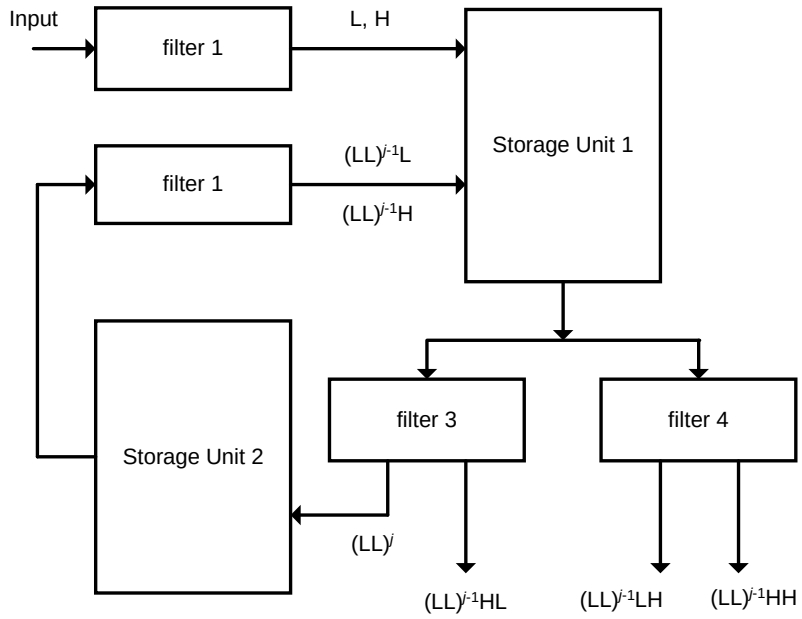


Figure 2.9 Two dimensional parallel filter architecture I of Chakrabarti *et al*

Input to the structure is in row-major order. Filter 1 computes L and H outputs in an interleaved fashion. Filter 2 first computes the output $(LL)^{j-1}L$ and N cycles later computes $(LL)^{j-1}H$. Filter 3 computes the outputs $(LL)^{j}$ and $(LL)^{j-1}HL$ along the columns. It first computes $(LL)^{j}$ and N cycles later computes $(LL)^{j-1}HL$. These outputs are stored in Storage 2. Similar to filter 3, filter 4 computes along the columns; it first computes $(LL)^{j-1}LH$ and N cycles later computes $(LL)^{j-1}HH$.

$L_{i,j}$ is scheduled at time $T + Ni + 2j$. $(LL)^{j}$ and $(LL)^{j-1}LH$ are scheduled to be computed at the same time and $(LL)^{j-1}L_{i,j}$ is scheduled to be computed T cycles after $(LL)^{j-1}_{i,2j}$ where T is the latency of the parallel filters in terms of clock cycles.

The scheduling of $LL^{a}_{i,j}$ is can be arranged in two ways depending on whether a reduced complexity and storage size or a reduced forward latency is desired. Scheduling algorithms are explained in detail in [37].

A modified version of the architecture in Figure 2.9 is shown in Figure 2.10. The modified architecture has been proposed for encoder-decoder systems where latency is an important issue. Outputs of all sub-bands are produced at the same time eliminating the extra buffering requirements at the decoder side.

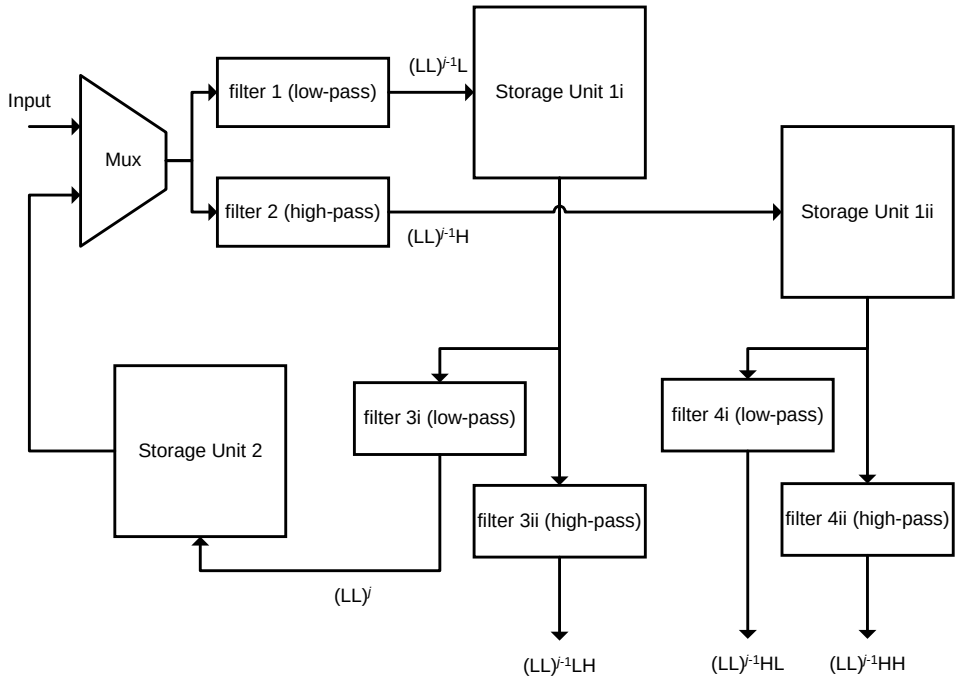


Figure 2.10 Two dimensional parallel filter architecture II of Chakrabarti *et al*

Storage units 1 and 2 of Figure 2.9 are of size $N/2+2LN\left(1-(1/2)^{J}\right)$ and $N\left(1-(1/2)^{J-1}\right)$ respectively. Sum of the size of storage units 1i and 1ii and the size of storage unit 2 of Figure 2.10 are $2LN\left(1-(1/2)^{J}\right)$ and $N\left(1-(1/2)^{J-1}\right)$ respectively. Both architectures have a computation time of approximately N^2 clock cycles.

A similar architecture to the ones explained above was proposed by Masud *et al* in [39]. It differs from the above architectures mainly in the scheduling method it uses. Architecture is presented as *semi-recursive pyramid algorithm architecture.*

The architecture is shown in Figure 2.11. It consists of two serial filters (filter 1 and 2) which operate on rows, a parallel filter (filter 3) which operates on columns, and two storage units, one of which stores L and H outputs while the other stores $(LL)^jL$ and $(LL)^jH$ outputs.

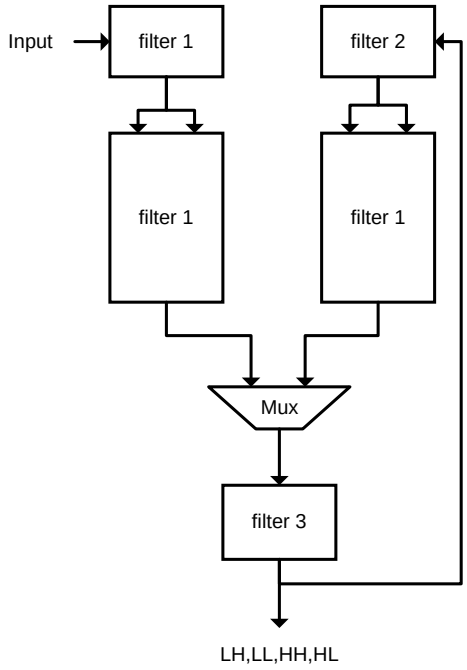


Figure 2.11 Semi-recursive pyramid algorithm architecture of Masud *et al*

The image is input in row-major order into the filter 1. Outputs of filter 1 are stored in memory 1. The rows in the memory are shifted by two row places. The parallel filter waits for a row to be complete in memory 1 and then begins to compute its outputs. The data needed for the computation of further levels are immediately filtered along the rows by filter 2 and then stored in memory 2. The parallel filter computes at twice the rate of filter 1. It computes the outputs of levels other than the first one as the filter 1 fills up the necessary rows for the first level. The scheduling algorithm of the architecture is shown in Figure 2.12. This scheduling results a more

simplified scheduling circuitry than the MRPA and RPA algorithms. And also boundary handling can be simplified.

Memory 1 is of size $2((L-1)\cdot\lfloor(N+L)/2\rfloor+1)$ and memory 2 is of size $2((L-1)\cdot\lfloor(Nj+L)/2\rfloor+1)$. The architecture computes the 2-D DWT of an $N\times N$ image in $3/2N^2$ clock cycles.

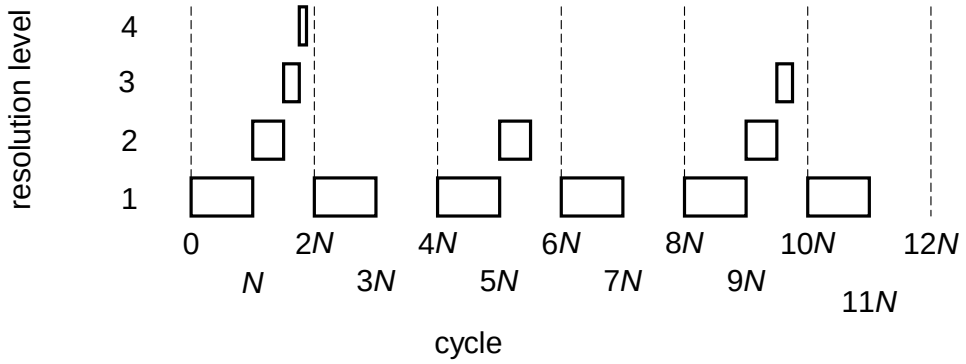


Figure 2.12 Scheduling of semi-recursive architecture

2.2.2.2.Non-Separable Filtering Architectures

A 2-D non-separable DWT architecture is presented in [33]. The architecture implements the MRPA algorithm in 2-D (ie higher level computations are interspersed between the first level computations). Figure 2.13 shows block diagram of the non-separable parallel filter architecture.

Each parallel filter consists of L^2 multipliers and a tree of (L^2-1) adders to add the products. The storage unit consists of J serial-in parallel-out shift register units, where the jth unit consists of 2-D array of storage cells of size L x $N/2^{j-1}$. Input data

is shifted into the storage units in rows. When a row is filled up data is shifted up one row. The architecture needs $2L^2$ multipliers, $2(L^2\text{-}1)$ adders, and $2NL$ storage cells. Computation time in terms of clock cycles for an image size of $N\times N$ is N^2.

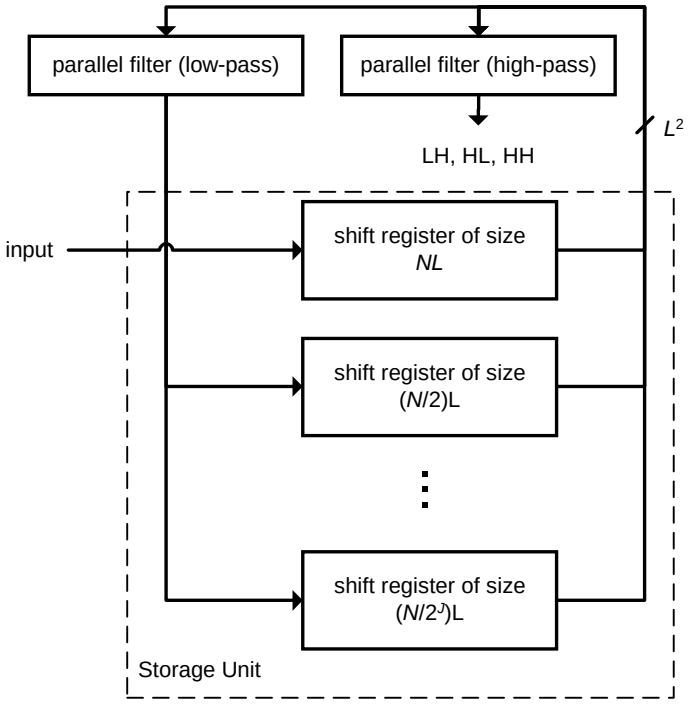


Figure 2.13 Non-separable filter architecture

This architecture results in a complex routing, however it is capable of applying non-separable filtering.

2.2.2.3.Systolic-Parallel Architectures

Viswanath *et al* have proposed a systolic-parallel architecture which implements the RPA in 2-D in [32]. The architecture consists of a systolic filter, a parallel filter and a systolic storage unit. The block diagram of the architecture is given in Figure 2.14. The systolic filter handles the filtering along the horizontal direction while the parallel filters handle the filtering along the vertical direction.

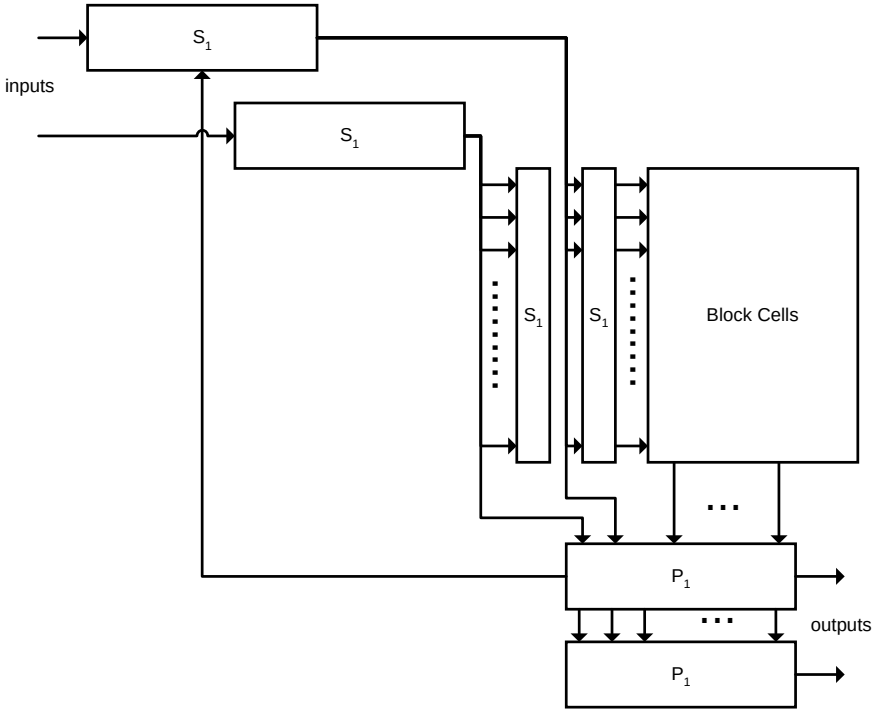


Figure 2.14 Systolic-parallel architecture

Two consecutive rows of the input frame are fed into S_1 and S_2 . The data of the routing network will come to S_1 from parallel filter P_1. The outputs of filters S_1 and S_2 are fed into the holding cells which shift their contents into the block cells once in every $2N$ cycles. The outputs in the block cells are stored in the same order as the output scheme of 1-D RPA. The filters P_1 and P_2 compute 4 rows over $2N$ cycles. These four rows constitute four outputs at that level. One of them is fed to S_1 . The parallel filters produce the outputs in the same order as the row filters and this is the required order for the row filter and routing network operation. The architecture computes 2-DWT of an N x N frame in N^2+N cycles. The storage is of size $2NL$.

2.2.2.4.Row-parallel RPA Architectures

An architecture which processes an entire row at the same time is proposed in [40]. This architecture does not use any MAC operators but bit-serial operation units

(PE's). It uses a scheduling scheme which can be interpreted as the 1-D RPA, in which the entire row is treated as a single pixel.

In Figure 2.15 the image data and wavelet coefficients are illustrated for a 3 level DWT with a filter length of 3 and a row size of 8 pixels. 8 PE's are required for the computation of 8-point DWT. The dash line illustrates the boundary of the data. Shaded coefficients are the ones required for the computation of the next level. Each PE computes one wavelet coefficient which resides in the corresponding column.(ie. Each column represents the computations of the corresponding PE) The data outside the boundary are obtained by mirror-extension.

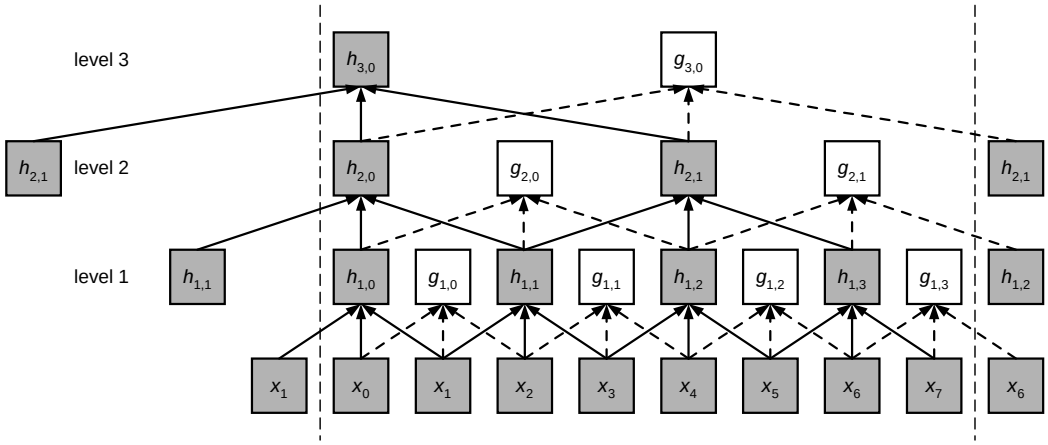


Figure 2.15 Row-parallel processing for three levels and image size of 8

Figure 2.16 shows the row-parallel processing architecture. The architecture has five main components: *N* number of PE's, a routing network, memory cells, the addressing elements and a controller. Input row to the structure is received and stored in memory. When the entire row has been received, the routing network reads the row and dispatches it to the *N* PE's. After the row operation is completed, column operation takes place. After the row and column operations LL band is stored in memory for higher levels of computation and other three bands are output. Higher bands computations are interspersed between the first bands computation as

in RPA algorithm. The method makes boundary handling much simpler than the architectures described earlier. Programming filter coefficients is also simplified.

The architecture uses $4N$ full-adders and memory of size $(L+1)NJ$ and no MAC operators. The computation time for an image size of $N \times N$ is ,as reported in [40], approximately N^2+N clock cycles.

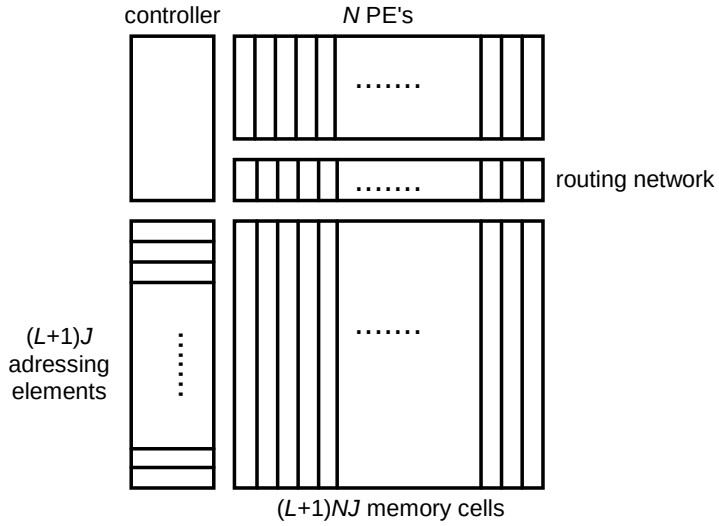
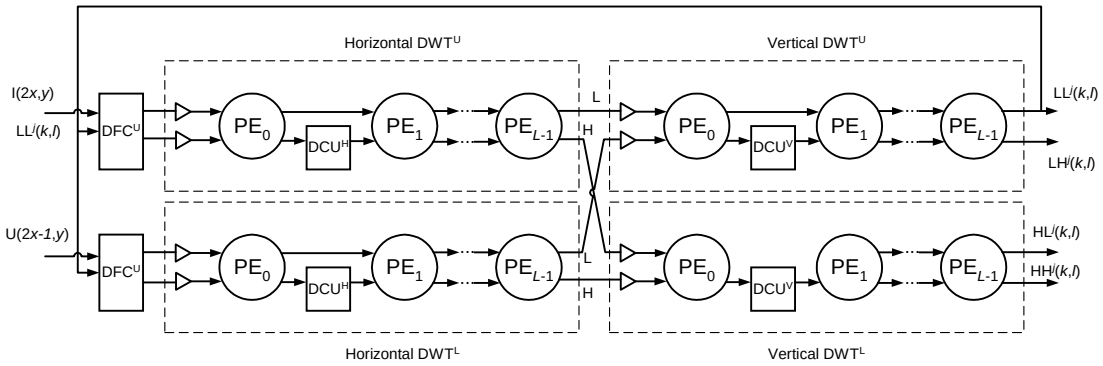


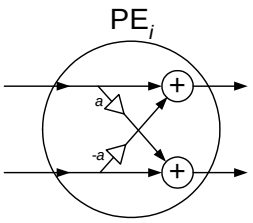
Figure 2.16 Row-parallel architecture

2.2.2.5.Lattice Architecture

Another architecture which has lattice structure filters is proposed in [41]. It is based on the extension of the lattice structure based 1-D DWT architecture proposed in [29]. Figure 2.17 illustrates the architecture, which consists of $4L$ processing elements (PE), two data format converters (DFC), and $4L$ delay control units (DCU). The architecture implements *paraunitary QMF factorization*. The theory behind this type of factorization is beyond the scope of this work, but for a more detailed discussion one can refer to [10].



(a)



(b)

Figure 2.17 (a) Lattice architecture. (b) Processing element.

In summary, the architecture accepts data from two rows at a time, while output rows of each level are scheduled row by row. Higher levels' rows are interspersed between the first level's rows.

The computation time for this method is $N^2/2$ since the architecture computes two rows at a time. The memory requirement is $2L(N+J)$. $8L$ multipliers and $8L$ adders are used.

2.2.2.6.Level-by-level transforming

A level-by-level transforming architecture is proposed in [42]. The block diagram of the architecture is shown in Figure 2.18. The architecture includes a transform

module which handles both horizontal and vertical filtering operations, a RAM module of size $N^2/4$, an address generator and a multiplexor. The computation scheme is as follows: In the first-level decomposition, the input is fed to the transform module, and the outputs LL, LH, HL and HH are generated. LL is stored in RAM in order to compute the second level outputs. After the completion of the first level, the data stored in RAM is fed to the transform module in order to compute LL^1 , LH^1 , HL^1 and HH^1.This procedure is repeated until the *J*th level computations. This scheme provides a regular output flow.

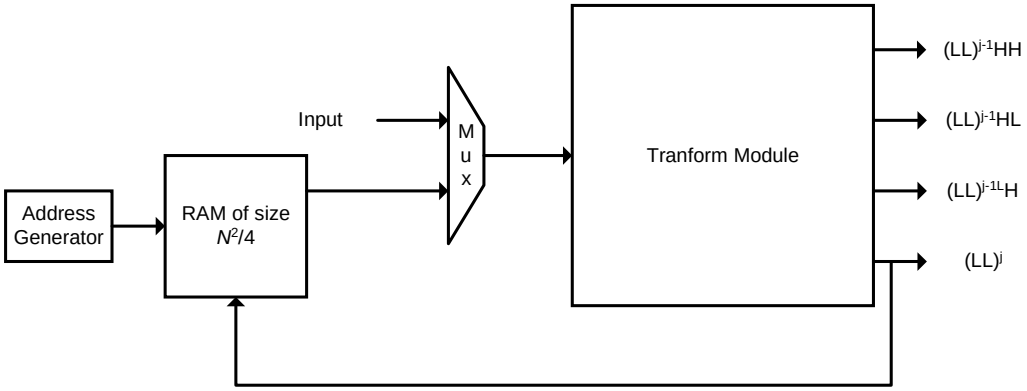


Figure 2.18 Level-by-level transforming architecture

The proposed architecture employs a polyphase decomposition technique in which the filter coefficients are decomposed into even-ordered and odd-ordered parts. During even clock cycles the input sample is fed to the odd-part and multiplied with the odd-ordered coefficients, during odd cycles the input sample is fed to the even-part and multiplied with the even-ordered coefficients as shown in Figure 2.19. The results from the two parts are summed up and output. This provides the internal clock of the architecture to be half the sampling rate. Hence the architecture computes 2-DWT of an $N\times N$ frame in $N^2/2$-$0.67N^2$ cycles depending on the J value. The total memory requirement for the architecture is $N^2/4+LN+L$.

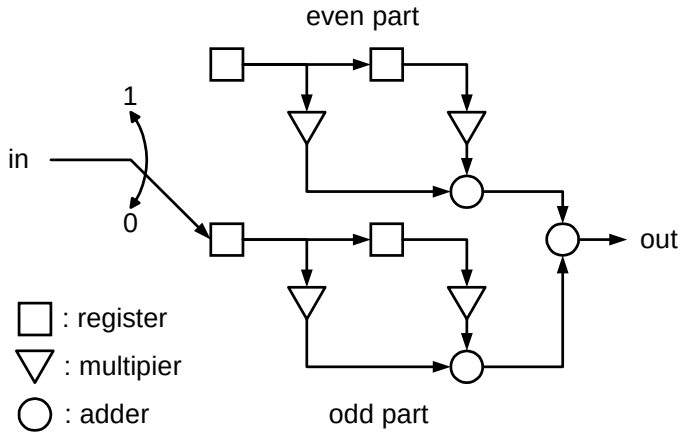


Figure 2.19 Splitted signal implementation

2.2.2.7.Quadri-filter

Two different 2-D DWT architectures employing "quadri-filter" blocks, which are suitable for non-separable 2-D filtering as well as separable filtering are presented in [45]. The qaudri-filter described in [45] is a modified type of 2-D convolver which splits the 2-D computation into several 1-D computations [46]. The Figure 2.20 illustrates a 2-D convolver having (*L*x*M*) taps. Each 1-D convolver C_i has *M* taps corresponding to the *i*th row of the 2-D filter. The row-adder computes the simultaneous sums of each partial result.

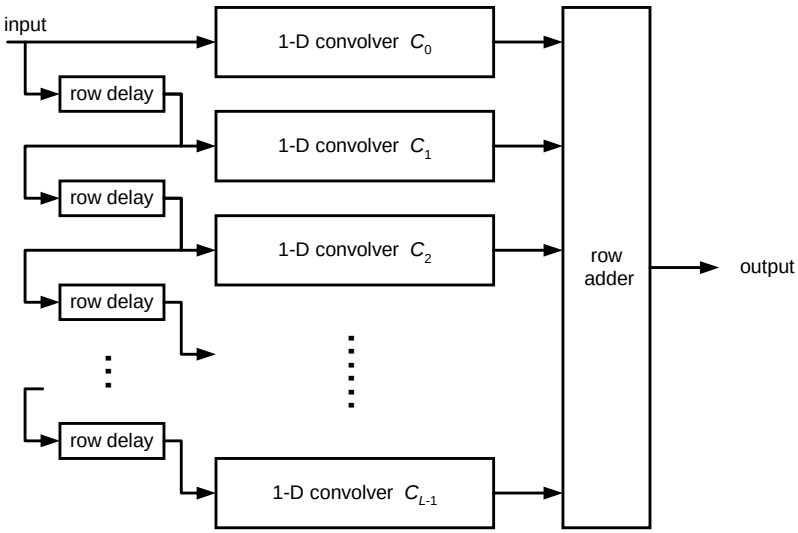


Figure 2.20 2-D convolver having $L \times M$ taps

Figure 2.21 illustrates the "Quadri-filter" block. "Quadri-filter" is capable of down-sampling and interleaved computation which are employed in DWT. The pipe of row-delays of 4.19 are split into two distinct pipes working in parallel. In this way, the even-ordered and odd-ordered rows can be fed simultaneously, thereby achieving down-sampling along rows. The processors P_i replace the 1-D convolvers of 4.19, and handle the computations of partial results $ll_{\{i\},m,n}$, $lh_{\{i\},m,n}$, $hl_{\{i\},m,n}$, $hh_{\{i\},m,n}$ $m,n=0,1..N/2^j$ in an interleaved fashion.

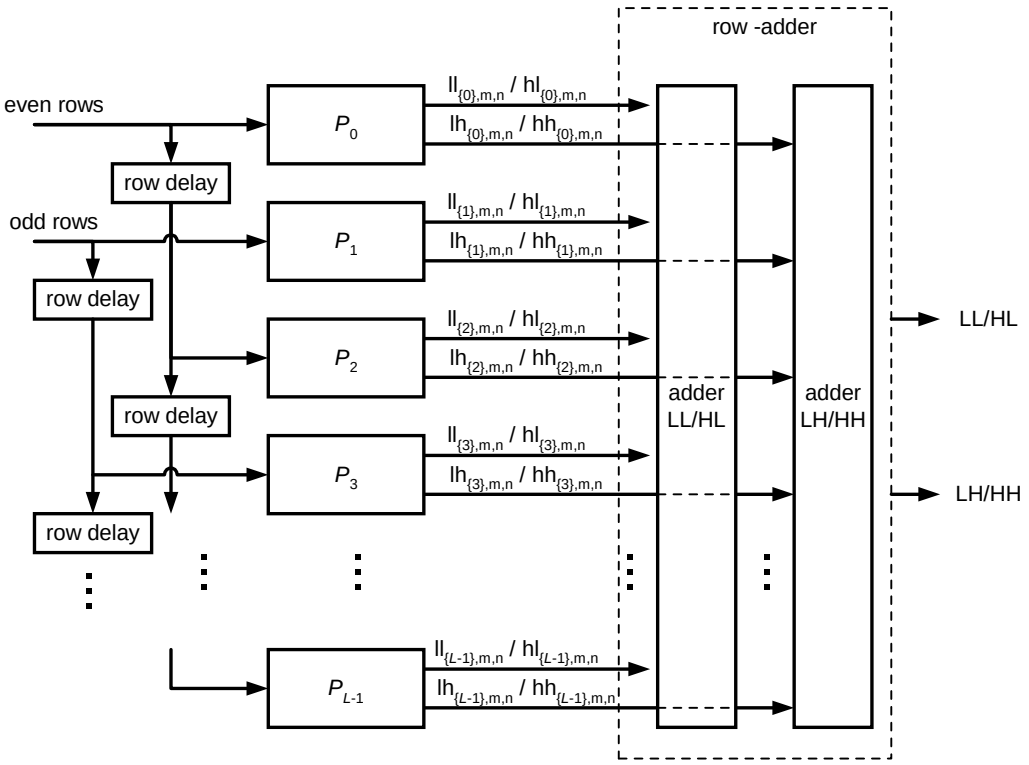


Figure 2.21 Quadri-filter structure

One of the two architectures proposed in [45] is a folded architecture which implements a MRPA-like scheduling algorithm as shown in Figure 2.22. The quadri-filter used in the folded implementation has J distinct row-delay pipe-lines. The data is fed to these pipe-lines through a demux and P_i's are fed through multiplexors which decide on the j value.

The other architecture is the pipe-lined approach which has as many processing units as the levels of decomposition as shown in Figure 2.23. In a pipe-lined 2-D DWT, the sub-band coefficients $lh^j_{\{i\},m,n}$, $hl^j_{\{i\},m,n}$ and $hh^j_{\{i\},m,n}$ are generated in slice j of the pipe-line and directly output. $ll^j_{\{i\},m,n}$ sub-band generated at slice j is fed into slice j+1. The quadri-filter blocks at each level differs from others in the size of the input it processes; the filter at level j processes an input of size $N/2^{j-1} \times N/2^{j-1}$. Note that the $lh^j_{\{i\},m,n}$ and $ll^j_{\{i\},m,n}$ outputs are generated in an interleaved fashion. Therefore, an adapter is required to handle the separation of sub-bands. The adaptor

also handles splitting the even-ordered and odd-ordered rows of the input and feeding them in parallel to the next level.

Both architectures have a memory requirement of $2LN$. The folded architecture has a computation time of approximately $2N^2/3$ clock cycles, where the pipe-lined architecture has a computation time of approximately $N^2/2$ clock cycles. The strength of the pipe-lined approach is that it requires more simpler control and routing circuitry compared to the folded architectures.

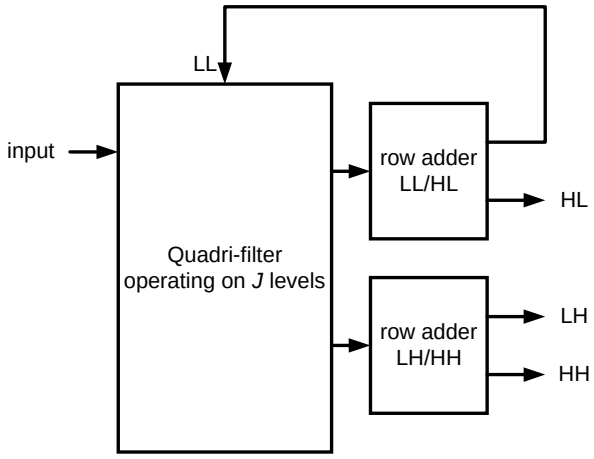


Figure 2.22 Folded architecture with quadri-filters

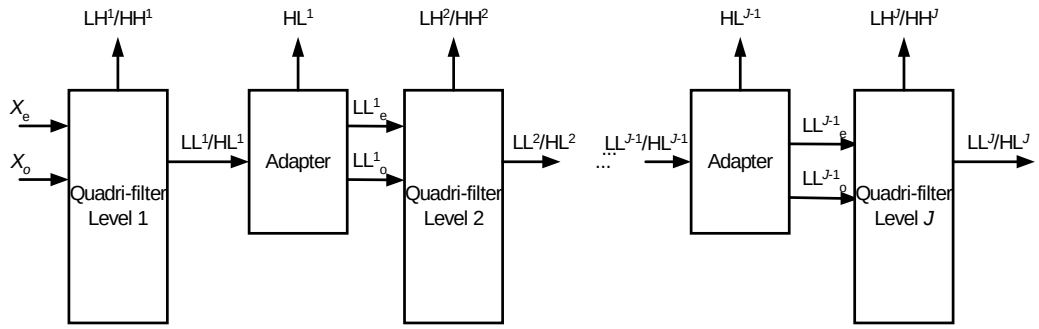


Figure 2.23 Pipe-lined architecture with quadri-filters

It is reported that in many practical cases, pipe-lined architectures are simpler than the conventional MRPA-based devices [45]. Hardware cost of pipe-lined architecture for the cases examined in [45] except {J>8, L=8, N=512} and {J>6, L=12, N=512} is smaller than the parallel filter architectures of [37]. It should be noted that even though J can be any integer not exceeding $\log_2(N)$, in many practical applications values of J greater than 4 or 5 (with N=512) and 3 or 4 (with N=256,128) produce little to no added benefits.

2.3 Summary and Comparisons on Mallat Tree Architectures

In this section the 2-D discrete wavelet transform architectures which have been discussed in previous sections will be summarized and compared from an FPGA implementation perspective. Specifically, the feasibility of realizing such architectures in FPGAs and suitability of these architectures for specific applications such as JPEG 2000 image compression standard [19] or tile-based processing of large image data, will be discussed.

Table 2.1 Comparisons of 2-D DWT Architectures (I)

Architecture	Storage Size (*B* bits)	Storage Size Bound	Suitability to RAM appl.	Computation Time (ccs)
I. Direct Approach [32]	N^2	$O(N^2)$	yes	$4N^2$
II. SIMD Architecture [33]	$2N^2$	$O(N^2)$	no	$4JL$
III. Parallel Filter 1 [37]	$\approx 3/2\ N+2LN$	$O(LN)$	yes	$\approx N^2$
IV. Parallel Filter 2 [37]	$\approx\ N+2LN$	$O(LN)$	yes	$\approx N^2$
V. Masud [39]	$\approx 2NL$	$O(LN)$	yes	$3/2\ N^2$
VI. Non-Seperable [33]	$2NL$	$O(LN)$	yes	N^2
VII. Systolic-parallel [32]	$2NL$	$O(LN)$	no	N^2+N
VIII. Row-parallel [40]	$(L+1)NJ$	$O(LNJ)$	no	$\approx N^2+N$
IX. Lattice Architecture [41]	$2L(N+J)$	$O(LN)$	no	$N^2/2$
X. Level-by-level [42]	$N^2/4+LN+L$	$O(N^2)$	yes	$N^2/2 - 0.67N^2$
XII. Semi-recursive [44]	$\approx N^2$	$O(N^2)$	yes	$4N^2/3$
XIII. Quadri-filter folded [45]	$2NL$	$O(LN)$	no	$2/3\ N^2$
XIV. Quadri-Filter pipe-lined [45]	$2NL$	$O(LN)$	no	$N^2/2$

FPGA designs are classified as ASIC designs; most of the design considerations for ASIC implementations are common for FPGAs and custom ICs. There are, however, several design differences depending on the specific application.

Memory is one of the prime issues for an FPGA application. The limited storage resources of FPGAs compared to which custom ICs offer, may make an FPGA implementation infeasible despite its suitability for a custom IC. Typical resources including both registers and on-chip block RAMs are given in [47]. The limited number of resources forces designer to concentrate on reduced memory implementations of the wavelet transform.

Table 2.1 shows the storage requirement and computation times of various implementations. The architectures can be roughly divided into two groups one of which has a storage requirement bounded with $O(LN)$ or $O(LJN)$ while the other group has a storage requirement bounded with $O(N^2)$. It is clearly preferable to design an architecture which is $O(LN)$ bounded in storage for reduced memory applications rather than an architecture which is $O(N^2)$ or $O(JN)$ bounded. For fast

applications, architectures having computation times independent of *N* may be preferred.

For transforming large images on FPGA, architectures of type I,II,X and XII are not feasible, since they require large sizes of storage. Even if an architecture with a smaller memory requirement is chosen, the required storage still may not be accommodated in the distributed registers available in FPGA devices [47-48], and the use of RAM blocks provided in such devices may be necessary. This also provides a considerable reduction in circuit complexity and routing. However not every type of architecture is suitable for RAM employment. The suitability of an architecture to use RAM blocks is given in Table 2.1.

Table 2.2 Comparisons of 2-D DWT Architectures (II)

Architecture	**Routing**	**Control/ Scheduling Complexity**
I. Direct Approach [32]	simple	simple
II. SIMD Architecture [33]	complex	complex
III. Parallel Filter 1 [37]	moderate	moderate
IV. Parallel Filter 2 [37]	moderate	moderate
V. Masud [39]	moderate	moderate
VI. Non-Seperable [33]	complex	complex
VII. Systolic-parallel [32]	complex	complex
VIII. Row-parallel [40]	complex	complex
IX. Lattice Architecture [41]	complex	complex
X. Level-by-level [42]	moderate	simple
XII. Semi-recursive [44]	simple	moderate
XIII. Quadri-filter folded [45]	complex	complex
XIV. Quadri-Filter pipe-lined [45]	moderate	simple

Routing and control complexity is another problem for FPGA devices. Large multiplexors bring complex routing and consumption of precious logic function generators [47]. Table 2.2 tabulates routing and control complexity of different architectures. Systolic, SIMD array and lattice architectures like II, VII, VIII, IX

have excessively large routing due to control and multiplexing/directing logic among the processing cells. Therefore although implementing those architectures on custom IC have many benefits, they are not suitable for FPGAs.

Folded architectures aim to minimize the number of filter modules by feeding multiple levels' outputs to a single filter module. This increases the utilization of the filters however brings extra multiplexing of intermediate outputs and the need for a scheduling circuitry to the system. Folded architectures like III-IX XIII require complex circuitry to schedule the output computation. For most applications N is much greater than L. This means that the area requirements for storage are the dominant design consideration and not the number of MAC units. Moreover for J values up to a certain number, folded architectures consume more area. [45].

Bus routing is another important issue. Internal RAM blocks may be spread over the entire FPGA device and it may require complex routing if a number of distant blocks are to be connected to the same data bus. Most of the RPA-like architectures have central control blocks which handle the data access by applying a harder strategy unlike architectures such as XIV, which handle data access of each level independently.

Table 2.3 Comparisons of 2-D DWT Architectures (III)

Architecture	Scalability	Programmable	Non-sep. Filtering	Frame Sequence Processing
I. Direct Approach [32]	Difficult	No	No	No
II. SIMD Architecture [33]	Easy	Yes	No	No
III. Parallel Filter 1 [37]	Difficult	No	No	Yes
IV. Parallel Filter 2 [37]	Difficult	No	No	Yes
V. Masud [39]	Moderate	No	No	Yes
VI. Non-Seperable [33]	Difficult	No	Yes	Yes
VII. Systolic-parallel [32]	Difficult	No	No	Yes
VIII. Row-parallel [40]	Easy	Yes	Yes	Yes
IX. Lattice Architecture [41]	Difficult	No	No	Yes
X. Level-by-level [42]	Moderate	No	No	*No*
XII. Semi-recursive [44]	Moderate	No	No	No
XIII. Quadri-filter folded [45]	Complex	No	Yes	Yes
XIV. Quadri-Filter pipe-lined [45]	Moderate	No	Yes	Yes

Scalability and coefficient programmability is another issue in DWT architectures. Some architectures are aimed to be generic architectures which are designed independent of the filter size and coefficients and can be adapted to different filters. However, some architectures are tuned for a single filter like in [36]. SIMD architectures like II and VIII are scalable and can easiliy be adopted to different filters. Systolic parallel architecture (VII) can also be expanded by some modifications on hardware. MRPA architectures and non-seperable filter architectures, on the other hand, are not easily expandable. In certain applications the architecture may be preferred to be programmable, i.e. user may program filters of different size and with different coefficients, or different image sizes. SIMD architectures like II and VIII are easy to expand and suitable for filter-programmable designs, however programming image size may be a problem for these architectures, while architectures having computation units independent of image size are much easier for image size programming.

Another point of concern is the dynamic range expansion. In DWT computations, the required precision increases with each level of sub-band decomposition. Since the folded architectures use a single computation module for all levels up, these architectures must accommodate the precision of the highest level *J*. Pipe-lined architectures, on the other hand, can be designed with different precision for each slice.

Some 2-D DWT filters are non-separable. Separable filtering approaches cannot be used for 2-DWT with non-separable filters. Among the architectures discussed above, only VI, VIII, XIII, XIV are capable of utilizing non-separable filtering.

Only a few of the studies in the literature address processing of the boundaries and usually zero padding is assumed. If the processes frame is a partition of a whole frame, zero padding may result degradation of the recovered image. Moreover, if perfect recovery is desired, symmetric extension must be applied at the boundaries. RPA based architectures are not efficiently modified to achieve boundary processing. This is because handling these boundaries requires extra control logic and modifications on scheduling procedures. However the row-parallel architecture (VIII), which has a SIMD array to process rows in parallel, has the inherent capability of applying symmetric extension. Architecture proposed by Masud *et al* (V) is reported to be easily modified concerning these boundaries if desired. It is reported to have reduced complexity in controllers used to process boundaries and thus to have a considerably higher efficiency in boundary processing compared to RPA-based architectures [39].

These architectures also differ in the reception of input data. In most applications input is received in raster scan. When an architecture receiving input in the same scan is used (as in I, III, IV, V, VI, VIII, X) additional data adapting circuitry is not needed. However architectures like VII, IX, XIII, XIV, some of which are capable of computing N^2 samples in cycles fewer than N^2, should be fed two rows at a time. This requires the additional row splitting circuitry.

Another concept is the ability to process consecutive frames. If the hardware is used repetitively to process a sequence of frames, the arrival of the frames may be another concern. The frames may be the partitions (tiles) of a bigger size image and there might be no gap between the reception of two consecutive frames. In this case some architectures may not have finished with a frame when the next one arrives. So these architectures may need extra input buffers to get rid of the early arrival of input data. I, II,X and XII are not suitable for processing frame sequences and require extra queue buffers.

CHAPTER 3

A REAL-TIME, LOW-LATENCY FPGA IMPLEMENTATION OF THE 2-D WAVELET TRANSFORM

This chapter presents the proposed architecture for a 2-D DWT processor in a multi-spectral imaging application environment. Figure 3.1 shows the block diagram of the 2-D DWT processor. The processor receives image data from *M* sources. In order to handle large sized images a pre-buffering and tiling method is proposed. Pixels are buffered into off-chip memory either after or before component transform (reversible color transform), and read back in re-ordered fashion (see Figure 3.1). The processor is capable of processing *P* image blocks in parallel.

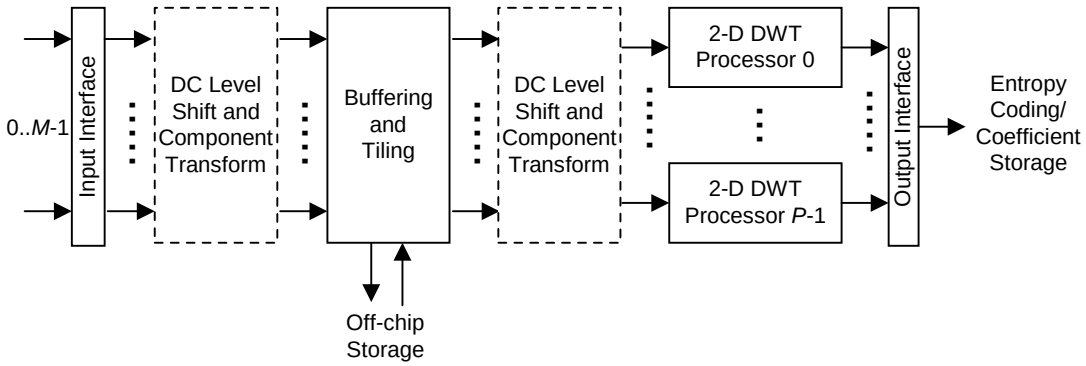


Figure 3.1 Block diagram of the DWT processor.

3.1 Input Data Stream Format and Notation

The data to the DWT processor is assumed to be received through M serial links each carrying image information from a camera of a specific spectrum. Each camera m ($m=0..M-1$) sends an image $\bar{I}_m(x,y)$ of n_1 x (n_2 + 2β) unsigned B-bit pixels in row-major order. For the reminder of the discussion every two dimensional signal $A(x,y)$ is called a *frame* and x denotes the vertical index and y denotes the horizontal index. The reception of data may be in quadrants since the image may have been partitioned into quadrants either internally by the CCD or by the camera electronics as shown in Figure 3.2. Both cases (with or without quadrants) are discussed whenever necessary.

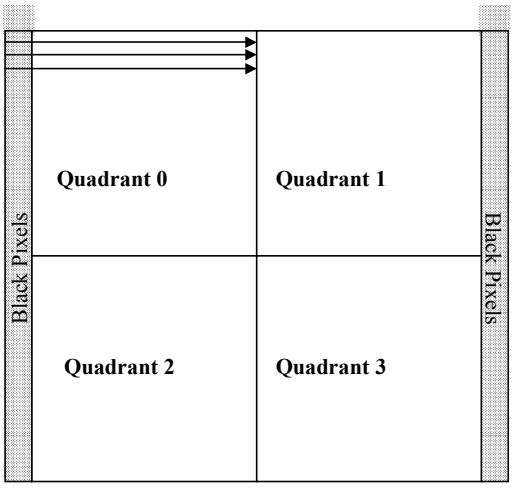


Figure 3.2 Image data acquisition with quadrants. On both sides of the image there are β pixels wide stripes which contain no information other than black pixels.

On both sides of the image (shaded areas in the Figure 3.2) there are β pixels wide stripes which contain no information other than black pixels (zeros). Therefore, data from these stripes should be ignored during acquisition.

The input interface block is designed in order to handle the physical layer considerations such as LVDS links etc. between the cameras and the DWT processor. It receives the serial bit stream, discards the black pixels and sends the images $I_m(x,y)$ of size n_1 x n_2 to rest of the hardware.

3.2 DC-Level Shift

Pixels from each camera are initially B bit unsigned where B is called "the bit depth" of the pixels As the JPEG 2000 standard suggests (see Figure 1.9) an offset of 2^{B-1} should be subtracted from the pixel values. After DC level shift, representation of the values become B-bits signed 2's complement. (i.e. $-2^{B-1} \leq I < 2^{B-1}$).

The motivation for the DC-level shift is that all sub-band samples produced by the DWT other than LL sub-bands involve high-pass filtering and hence have a symmetric distribution about 0. With the level offset samples can be assumed as signed with a bit depth of B+1, however this would increase the bit depths of computation. [10]

DC-level shift can be done just as the pixels are received from the input interface. Subtracting 2^{B-1} from a B-bit value in 2's complement arithmetic is equivalent to inverting the MSB of the value.

3.3 Color Transform

According to the JPEG 2000 standard, component transform is optional. However it is a known fact that the transformed components lead to better compression performance and shorter coding time than the RGB components [49].

The transform converts the RGB data into an "opponent" color representation, with a luminance channel, and two color difference channels. This has the effect of exploiting some of the redundancy between the original color components. In particular, color difference components commonly account for less than 20% of the bits used to compress a color image [10].

For this reason reversible color transformation is essential in our implementation. Reversible color transform employs a matrix operation on some components of the image (see (3.1)). The transform requires at least 3 components. The remaining components are left unchanged. For the cases $P<3$ this transform should be performed before the *tiling* step because after tiling only P number of components are available at the same time (see Figure 3.1). The component transformed data can be explained as follows:

$$X_0(x,y) = \left\lfloor \frac{I_0(x,y) + 2I_1(x,y) + I_2(x,y)}{4} \right\rfloor \tag{3.1a}$$

$$X_1(x,y) = I_2(x,y) - I_1(x,y) \tag{3.1b}$$

$$X_2(x,y) = I_0(x,y) - I_1(x,y) \tag{3.1c}$$

where I_0 , I_1 and I_2 are assumed to be red, green and blue components respectively.

For an architecture having P less than 3, computation of the transform components and writing them to off-chip memory presents a problem, which is the expansion of dynamic range in some components. It is apparent in (3.1) that the original components I_0, I_1, I_2 and I_3 are of B bits depth, but the new components X_0 , X_1 and X_2 are of B, B+1 and B+1 bits depth respectively (X_m is the same as I_m for $4 \le m < M$). Storing B+1 bit values is usually not feasible for a low memory hardware since the excess bits may require extra words or higher bandwidth to transmit them or large buffers in order to be rearranged in multiples of B bit words. In such cases the expanded components should be either;

i. quantized: least significant bit should be omitted as follows:

$$\breve{X}_k(x,y) = \left\lfloor \frac{X_k(x,y)}{2} \right\rfloor \tag{3.2}$$

for k=1,2

or,

ii. clamped: clamping the values to B-bit signed range ([-2^{B-1} , 2^{B-1}-1 as follows:

$$\breve{X}_k(x,y)=\begin{cases} -2^{B-1} & \text{for } X_k(x,y)<-2^{B-1}, \\ X_k(x,y) & \text{for } -2^{B-1}\le X_k(x,y)<2^{B-1}, \\ 2^{B-1}-1 & \text{for } X_k(x,y)\ge 2^{B-1} \end{cases} \tag{3.3}$$

for k=1,2.

Computation of X_1 and X_2 involves subtraction of original components. Although the dynamic range is increased one bit after subtraction theoretically, it actually reduces the dynamic range on correlated data such as natural images. Experiments show that clamping gives better results for satellite images.

3.4 Lifting Implementation

Consider the lifting structure in Figure 1.8 consisting of a series of filter-and-add steps. A possible way of row-based implementation of such a structure is to perform one step at a time, that is, each row is first processed in step one and the intermediate result is stored and then this result is fed to the second step and so on. However, this method requires a local memory which is equal to the row length. Furthermore, if the filtering is along columns in which the memory required for a single tap is equal to the row length, the whole frame should be stored. This method is not feasible for a low memory hardware implementation.

Another way is to process each step in cascaded fashion. In this case the intermediate results are immediately fed to the next step. The prediction and update filters $\widehat{P}_k(z)$ and $\widehat{U}_k(z)$ in Figure 1.8 may not be –which is usually the case- causal. For a non-causal system access to the entire signal is required. However the signal samples may be received one by one or a low memory application may not accommodate such a large number of samples. Therefore, causal versions of predict and update filters may have to be used. In [50] and [51] an efficient memory usage

for row-based lifting implementations is presented. A brief summary of the results in [50] is presented below.

Both $\widehat{P}_k(z)$ and $\widehat{U}_k(z)$ of Figure 1.8 may be thought as the summation of a causal and a purely anti-causal filter, such that:

$$\widehat{P}_k(z) = P_k^a(z) + P_k^c(z) \tag{3.4a}$$

$$\widehat{U}_k(z) = U_k^a(z) + U_k^c(z) \tag{3.4b}$$

where superscript *a* and *c* are used to denote the anti-causal and causal parts respectively. At this point the reader should be informed that the analysis in [50] assumes that the transfer function applied to the odd branch before the down-sampling is not z but z^{-1}, and therefore differs from the structure of Figure 1.8. Nevertheless, it does not affect our calculations.

It should be also noted that the notation used to discriminate between the forward and reverse filters in Section 1.2.2 is absent here. For consistency with the discussion in Section 1.2.2, the anti-causal and causal parts of the filters should actually be denoted by $\widehat{P}_k^a(z)$, $\widehat{U}_k^a(z)$ and $\widehat{P}_k^c(z)$, $\widehat{U}_k^c(z)$ respectively. Since our analysis here focuses only on forward path, a simpler notation is preferred for the present discussion.

The filters can be given as :

$$P_k^a(z) = p_{-l_k} z^{l_k} + p_{-l_k+1} z^{l_k} + \cdots + p_{-1} z \tag{3.5a}$$

$$P_k^c(z) = p_0 + p_1 z^{-1} + \cdots + p_{g_k-1} z^{-g_k+1} + p_{g_k} z^{-g_k} \tag{3.5b}$$

$$U_k^a(z) = u_{-m_k} z^{m_k} + u_{-m_k+1} z^{m_k} + \cdots + u_{-1} z \tag{3.5c}$$

$$U_k^c(z) = u_0 + u_1 z^{-1} + \cdots + u_{f_k-1} z^{-f_k+1} + u_{f_k} z^{-f_k} \tag{3.5d}$$

For prediction and update steps in the above equations we assume that:

$$k = 0,1,2\ldots N_{p-1} \tag{3.6a}$$

$$k = 0,1,2\ldots N_{u-1} \tag{3.6b}$$

respectively, and

$$l_k = g_k = 0 \quad \text{for} \quad k \neq 0,1,2\ldots N_{p-1} \tag{3.7a}$$

$$m_k = f_k = 0 \quad \text{for} \quad k \neq 0,1,2\ldots N_{u-1} \tag{3.7b}$$

In order to realize a non-causal system which has computations based on future samples, we have to buffer a proper amount of the signal until no future sample is required to compute the earliest output sample. Mathematically speaking, we use the *causal versions* of the filters instead of the original ones.

Let $\widetilde{P}_k(z)$ and $\widetilde{U}_k(z)$ be the causal versions of $\widehat{P}_k(z)$ and $\widehat{U}_k(z)$ of Figure 1.8, which can be obtained by multiplying the transfer functions by the least possible delays z^{-l_k} and z^{-m_k} respectively:

$$\widetilde{P}_k(z) = z^{-l_k} \cdot \widehat{P}_k(z) \tag{3.8a}$$

$$\widetilde{U}_k(z) = z^{-m_k} \cdot \widehat{U}_k(z) \tag{3.8b}$$

Figure 3.3 shows the realization of the structure with causal filters. It is apparent that the amount of delay applied to make the filter causal should also be applied to the unfiltered side.

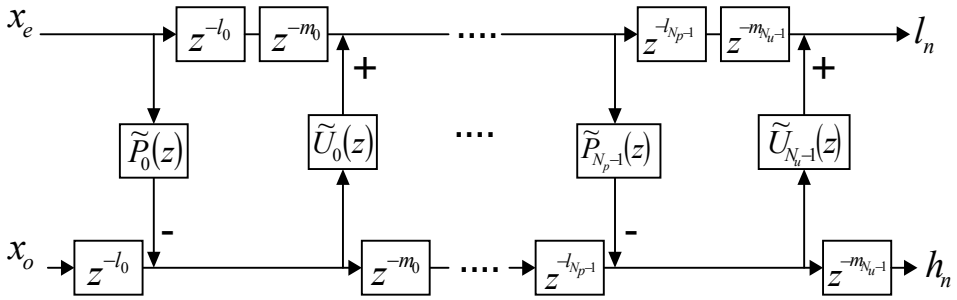


Figure 3.3 Lifting realization with causal filters.

Note that in Figure 3.3 the delay lines are repeated at the filtered channel side in order to synchronize the outputs of the stage. The taps for the anti-causal part of the filters can be provided from these delay lines and thus no extra memory is needed for calculation of $z^{-l_k}P_k^a(z)$ and $z^{-m_k}U_k^a(z)$ outputs. To compute the outputs of the causal parts of the filters, $z^{-l_k}P_k^c(z)$ and $z^{-m_k}U_k^c(z)$, we need more g_k and f_k delay elements. The required delay line can be provided by applying z^{-g_k} and z^{-f_k} to the branches placed at the outputs of prediction and update steps respectively as shown in Figure 3.4.. In figures 3.3 through 3.6, square boxes represent delay elements.

The delays z^{-g_k} and z^{-f_k} can be combined with the delays of the next lifting step (the larger ones absorb the smaller ones) which results in the structure in Figure 3.5.

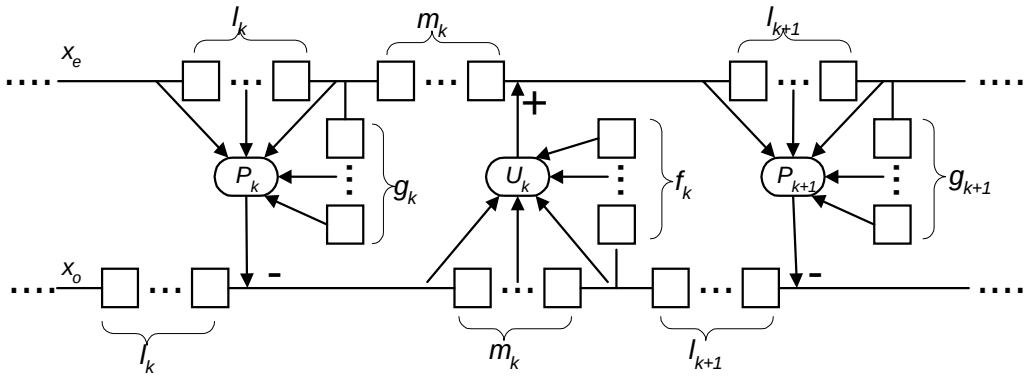


Figure 3.4 Delay lines of lifting implementation

We define γ_k and ϕ_k as :

$$\phi_k = \max\{g_k, m_k\} \tag{3.9a}$$

$$\gamma_k = \max\{l_k, f_{k-1}\} \tag{3.9b}$$

Then, the total memory required to implement filtering is found to be:

$$\mu = \sum_{k=0}^{N}(\phi_k + m_k) + \sum_{k=0}^{N}(\gamma_k + l_k) \tag{3.10}$$

where,

$$N = \max\{N_u, N_p\} \tag{3.11}$$

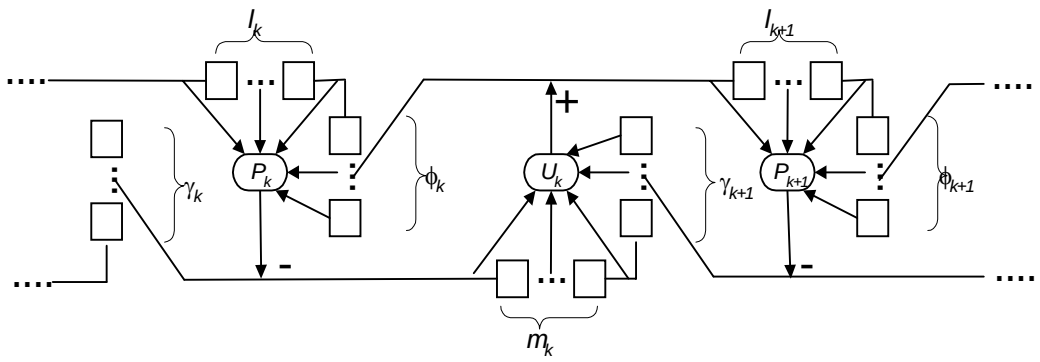


Figure 3.5 Modified delay line structure

Clarification of some of the more obscure points in these calculations may be in order:

Let a delay of z^{-1} be inserted into both the even and odd branches just after the down-samplers (the leftmost side in Figure 3.3). The introduced delay at the even branch can be reflected to the input side of the down-sampler as z^{-2} . The one at the odd side can be merged with z^{-l_0}, and all the delays along the path can be rearranged so that each delay block donates to or accepts from the adjacent block a delay of z^{-1}. Cancelling the delays at the input side we end up with a system similar to Figure 3.3. . Although we have the same predict and update filters, l_k's and m_k's have been modified as follows:

$$l'_k = l_k + 1 \tag{3.12a}$$

$$g'_k = g_k - 1 \tag{3.12b}$$

$$m'_k = m_k - 1 \tag{3.12c}$$

$$f'_k = f_k + 1 \tag{3.12d}$$

$$\phi'_k = \max\{g'_k, m'_k\} \tag{3.12e}$$

$$\gamma'_k = \max\{l'_k, f'_{k-1}\} \tag{3.12f}$$

$$\mu' = \sum_{k=0}^{N}(\phi'_k + m'_k) + \sum_{k=0}^{N}(\gamma'_k + l'_k) \tag{3.12g}$$

As far as the values of the output samples are concerned, disregarding the delay introduced, we can say that "causality degree"s of the filters change depending on the type of input. For the one in Figure 3.3, predict filters are "more causal" and the update steps are "less causal", while, for the modified structure, update steps are "more causal" and the predict steps are "less causal". As an example the 5/3 filter of [15] which is used in our hardware can be given.

The predict and update steps of 5/3 filter are as follows:

$$P_0 = \frac{1}{2}(1\ 1), \qquad U_0 = \frac{1}{4}(1\ 1) \tag{3.13}$$

and

$$N_p = 1, \qquad N_u = 1 \tag{3.14}$$

For the input mode depicted in Figure 3.3, i.e. where odd samples lag even samples; l, m, γ, ϕ values are as follows:

$$l_0 = 0, \quad m_0 = 1, \quad g_0 = 1, \quad f_0 = 0, \quad \phi_0 = 1, \quad \gamma_0 = 0, \quad \gamma_1 = 0 \tag{3.15a}$$

$$\mu = l_0 + m_0 + \phi_0 + \gamma_0 + \gamma_1 = 2 \tag{3.15b}$$

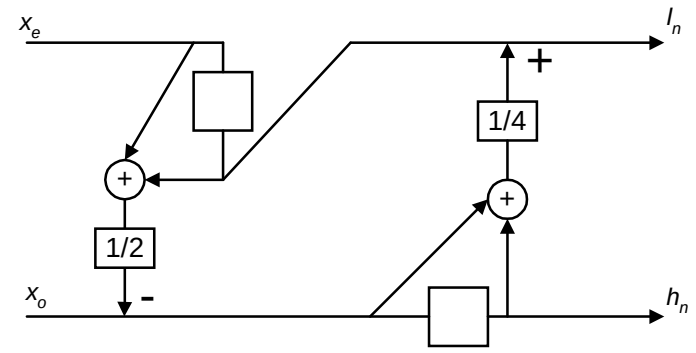
and for the modified input mode i.e where odd samples lead even samples; l, m, γ, ϕ values are as follows:

$$l_0' = 1, \quad m_0' = 0, \quad g_0' = 0, \quad f_0' = 1, \quad \phi_0' = 0, \quad \gamma_0' = 1, \quad \gamma_1' = 1 \tag{3.16a}$$

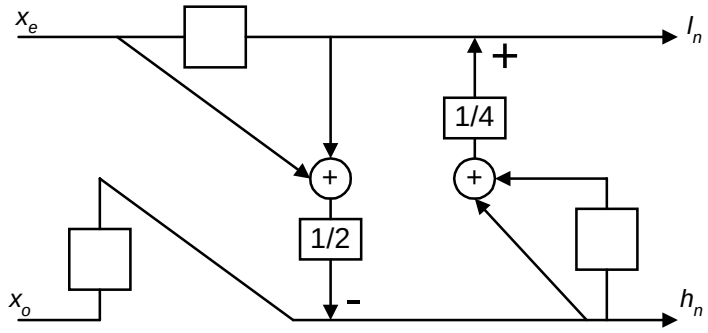
$$\mu' = l_0' + m_0' + \phi_0' + \gamma_0' + \gamma_1' = 3 \tag{3.16b}$$

Note that in (3.12) the delay required to split the signal into even and odd parts is not included. Furthermore, one more delay may be needed if the odd and even signals are to be combined at the end of the filtering.

Both structures for the 5/3 filter are shown in Figure 3.6a and Figure 3.6b.



(a)



(b)

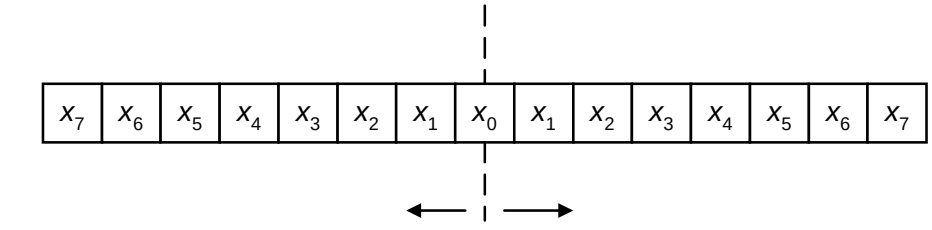
Figure 3.6 Lifting structure of 5/3 filtering. (a) Odd samples lag even samples, (b) Even samples lag odd samples.

From the above discussion it is apparent that the even-lagging system needs less storage than the odd-lagging system; therefore it is preferred. However the low-pass (even) outputs and high-pass (odd) outputs do not begin and end at the same time; odd outputs begin one sample later than the even outputs. For the purpose of parallel processing at the next stages of filtering, filters can be arranged so that one sample earlier version of the high-pass outputs which are already in f_{N_u-1} registers can be taken as outputs.

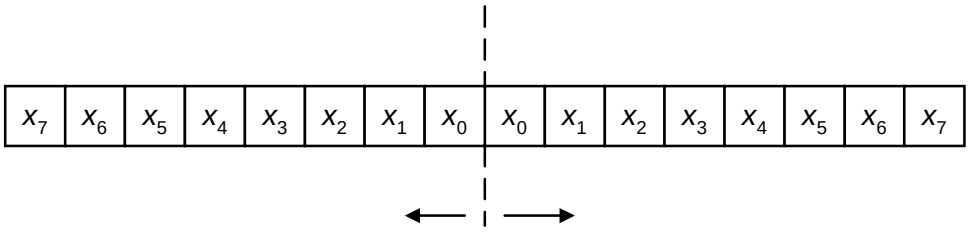
3.5 Symmetric Extension in Lifting Steps

The main purpose of using symmetric extension is to maintain the symmetry of the samples around the boundaries even after filtering. This is needed for perfect reconstruction. Symmetry of filtered coefficients is possible in the case where symmetric filter kernels are used [52]. In this thesis we consider only the odd-length filter kernels applied to even length signals. Other combinations of filter kernels and signal lengths are beyond the scope of this work.

For an even-length signal which is to be filtered with odd-length filters *whole point extension* should be applied at the boundaries (see Figure 3.7). One way of applying symmetric extension is to copy the required amount of beginning samples ahead of the signal and ending samples after the signal before feeding it to the filters. However, copying existing samples may constitute a problem in terms of memory, especially when filtering along the vertical direction. Since each delay buffer has the same length as a row, a number of rows equal to half the filter length should be stored.



(a)



(b)

Figure 3.7 Symmetric extensions: (a) Whole point (b) Half point

A better way of applying symmetric extension is to use the memory already available in the filters. The required extension of the signal, which is half the length of the filter, can be copied symmetrically as the signal is received in. At the beginning of the signal, while the incoming samples are shifted in from one end of the filter, they are also copied symmetrically to the other end. Filter buffers are filled up as the first output is to come out and thus it is computed with the valid values in the filter. At the end of the signal the samples inside the filter are fed back to the filter in order to compute the last output samples [50].

This method applies to not only convolution filters, but also to the lifting steps implementations as well [50]. Each lifting step can handle its own symmetric extension individually according to the beginning and ending of the signal it receives from the previous step. Figure 3.8 shows which types of symmetric extension should be applied to the odd and even parts of the even-length signal in order to achieve the equivalent whole-point extension which should be used in the convolution filter. Even samples are required to be whole-point extended at the beginning of the signal and half-point extended at the end of the signal, while odd samples are required to be half-point extended at the beginning and whole-point extended at the end.

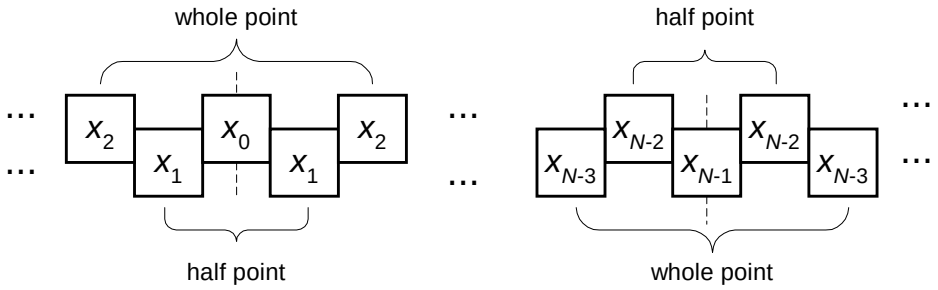


Figure 3.8 Application of symmetric extension to even-length signals.

With this method, the need of extra memory for storing copy samples is eliminated. However, a main drawback of such a system may be the added time required to complete the filtering due to copy samples:

Note that the filter buffers are not fully utilized at the beginning and ending periods of the signal in such a system. i.e. a useful output is not always produced for every shift of samples. This is not a problem for the processing of a single row/frame where the signal begins and end once. However, if a sequence of row/frames are to be processed with a fully utilized input transmission, this structure will lack the time to process the boundary data. The required gap in terms of row time between the input tiles can be calculated with a similar method as in (2.5).

In order to provide the full utilization of the filter, a modified structure is proposed, where instead of duplicating or feeding back the genuine samples at the boundaries, multiplexing the proper samples into the MAC operator is used. The idea is to emulate the symmetric extension at the input of the MAC operators. This corresponds to a particular application of *time/location varying filters* instead of fixed filters. Time varying filters for finite signals are explained in detail in [53] With this scheme, the filter buffers are only used to apply straight shifting on the received samples. This also provides a regular buffering pattern and simplifies the read back operation of the tiling module.

It is possible to construct a lifting circuit with the method explained above, which is equivalent to any convolution filtering with symmetric extension where even-length signals and symmetric odd-length filter kernels are used.

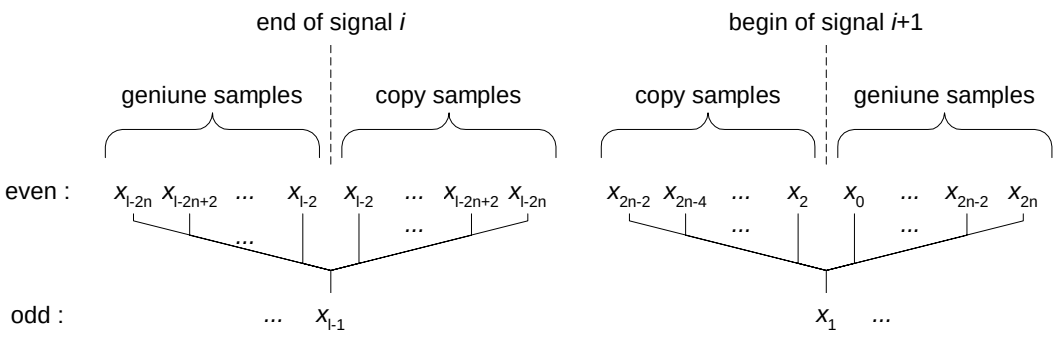


Figure 3.9 Symmetric extension applied to consecutive signals.

It was mentioned earlier that the required type of symmetric extension is whole-point for odd-length filters and even-length signals. Since lifting of odd-length symmetric filters lead to even-length symmetric lifting filters and even-length filters lead to odd-length symmetric filters [13], we always deal with lifting filters which are even-length. Let the filter length of such a filter be $2n$ and the length of the signal be l. As it is apparent from Figure 3.9, in order to compute the first odd output sample, n+1 genuine samples are needed while n genuine samples are needed to compute the last sample of the previous signal. Assuming that the copy samples can be obtained by multiplexing, at time t (where the last output of prior signal is released) the filter should be storing n+1 genuine samples from the prior signal, and at time t+1, (where the first output of the latter signal is released), it should be storing n genuine samples from the latter signal. This guarantees that the filter should at most –excluding the most recently received sample- store $(n+1)+n-1=2n$ samples at a given time. Figure 3.10 shows the buffer content at the joint of two consecutive signals where l=16 and $2n$=8. Main entries denote the index of the contained sample where superscript denotes the multiplexed data to the computation units. Shaded numbers belong to the prior signal.

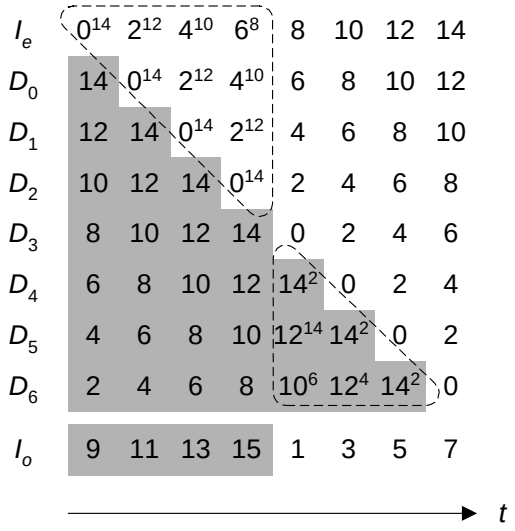


Figure 3.10 The buffer content at the joint of two consecutive signals where l=16 and $2n$=8.

With the added boundary handling properties, filter architectures become more complex; they involve state machines which interpret the row/tile beginning and ending signals and multiplexing circuitry, and hence diverge from being a combination of dummy tap delay lines and MAC operators. Multiplexing schemes, signal beginning/ending and memory access controls can be combined in a processing block called *control unit* and the module handling weighted sum computations can be referred to as *computation kernel.* The elements of a filter is shown in Figure 3.11.

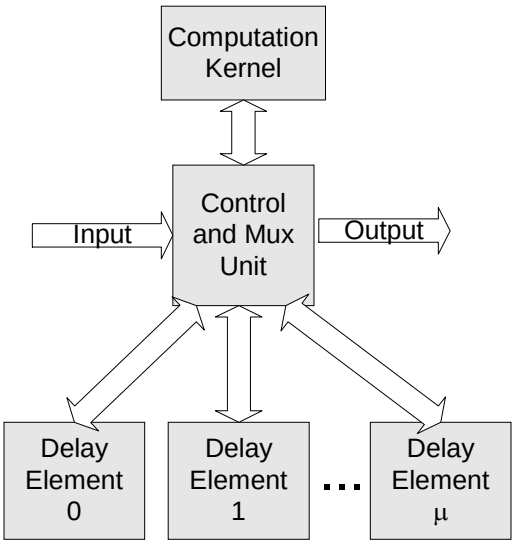


Figure 3.11 The elements of a filter architecture for lifting.

3.6 Tiling

In Section 2.3 it was explained that the memory size of most 2-D DWT architectures is dominated by the vertical filter requirement and is $\boldsymbol{O}(LN)$ bounded. Since the processing of $n_1 \times n_2$ size images would require an excessively large vertical filter, the original image is tiled into smaller $(n_1/k_1) \times (n_2/k_2)$ size tiles ($k_1, k_2 \in \mathbf{Z}$) as described in [54][10].

Although reducing the size of the vertical filter storage dramatically, tiling introduces the need to store a portion of the image locally before processing can begin. For an image of size $n_1 \times n_2$ and tile sizes of $(n_1/k_1) \times (n_2/k_2)$, a $(1/k_1)$ portion of the image should be accumulated before one tile can be formed. Tiles are therefore formed in groups of k_2 for each $n_1 \times n_2$ size image. Since the processing of any given group of tiles overlaps with the accumulation of the next group, a $(2/k_1)$ portion of each image must be stored locally at any given time. This also introduces a latency equal to the time elapsed to accumulate a $(1/k_1)$ portion of the image. Note that the maximum possible number of tiles to be fetched from storage

simultaneously is equal to the number of tiles in a group. Therefore more than $P=Mk_2$ number of parallel processing units are redundant.

The amount of data buffered before beginning computation (and hence the delay introduced due to buffering) in terms of pixels is:

$$S = \frac{Mn_1n_2}{k_1} \tag{3.17}$$

As explained above, the local storage size to perform tiling is 2*S*. The latency and local storage requirement can be reduced by half by receiving the images divided into four quadrants as shown in Figure 3.12. This, however, requires some minor modifications to the CCD camera read-out circuitry.

Each frame $\breve{X}_i$ is divided into tiles forming a tile matrix as shown in Figure 3.12. The tile $\breve{X}_i^{q,r,s}$, which is on the *r*th row and *s*th column of the *q*th quadrant (if quadrants are used) of the tile matrix, is assigned a global-tile index *g* and is equal to a portion π of a "global-tile", $\mathbf{T}_g$ (π=0..*P*-1). Since the hardware performs decoupled and parallel operations on $\overset{\pi}{T}_g$'s, instead of mentioning $\overset{\pi}{T}_g$'s individually, for simplicity we can assume the concatenation $\mathbf{T}_g$ as a vectorized form of $\overset{\pi}{T}_g$, which consists of $P\times B$ bits , such as :

$$\mathbf{T}_g \overset{\Delta}{=} \begin{pmatrix} \overset{0}{T}_g \\ \cdots\cdots \\ \overset{1}{T}_g \\ \cdots\cdots \\ \vdots \\ \cdots\cdots \\ \overset{P-1}{T}_g \end{pmatrix} \tag{3.18}$$

A notation similar to that of $\mathbf{T}_g$ is used for all intermediate and output frames in the remainder of this discussion.

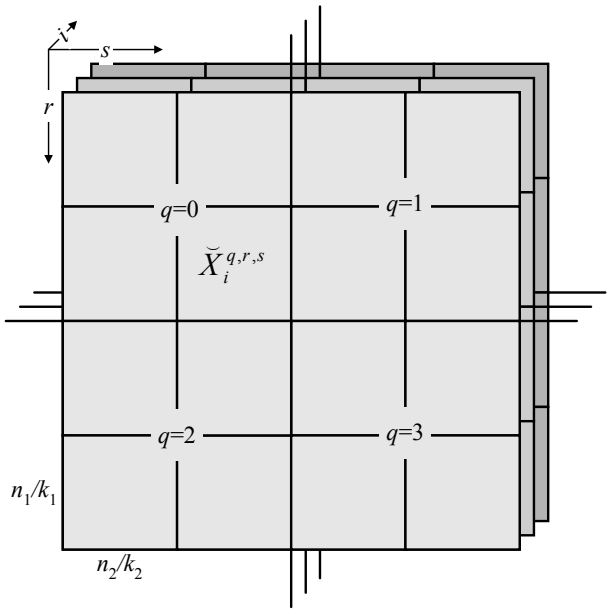


Figure 3.12 Application of tiling to the image components. Each frame $\breve{X}_i$ is divided into tiles forming a tile matrix. The tile $\breve{X}_i^{q,r,s}$, which is on the *r*th row and *s*th column of the *q*th quadrant (if quadrants are used) of the tile matrix, is assigned a global-tile index *g* and is equal to a portion π of a "global-tile", $\mathbf{T}_g$ (π=0..*P*-1).

For a frame size of $n_1 \times n_2$ and tile size of $(n_1/k_1) \times (n_2/k_2)$, with *M* spectral components and *P* parallel processing units, there are two cases:

i. without quadrants:

$$g = \left\lfloor \frac{k_2 Mr + Ms + i}{P} \right\rfloor \qquad (r = 0,1..k_2 - 1 \text{ and } s = 0,1..k_1 - 1) \tag{3.19a}$$

$$\breve{X}_i^{r,s}(x,y) = \overset{\pi}{T}_g(x,y) \qquad \text{for } k_2 Mr + Ms + i \equiv \pi \,(\text{mod}\, P) \tag{3.19b}$$

ii. with quadrants (note that k_1 and k_2 should be even) :

$$
\begin{aligned}
&g = \left\lfloor \frac{k_1 k_2 / 4 \cdot Mrq + k_2 / 2 \cdot Mr + Ms + i}{P} \right\rfloor \\
&\left(r = 0,1..k_2 - 1 \;\; s = 0,1..k_1 - 1 \;\; q = 0,1,2,3\right) \\
&\breve{X}_i^{q,r,s}(x,y) = \overset{\pi}{T}_g(x,y) \qquad \text{(3.19c)} \\
&\text{for } \; k_1 k_2 / 4 \cdot Mrq + k_2 / 2 \cdot Mr + Ms + i \equiv \pi \,(\mathrm{mod}\, P)
\end{aligned}
$$

Tiles are processed in the *global-tile order* : $\mathbf{T}_0$,…, $\mathbf{T}_g$, $\mathbf{T}_{g+1}$,… $\mathbf{T}_{Z-1}$, where Z is the number of global-tiles, and it is equal to :

$$
Z = \left\lceil \frac{k_1 k_2 M}{P} \right\rceil \qquad \text{(3.20)}
$$

After S pixels have been received the reading of tiled image data can begin reading and writing is continued simultaneously until the entire image has been received. The direction of reading is either from top to bottom first and from left to right (column-major order) or from left to right first and from top to bottom (row-major order). Which direction of read back results in a smaller memory requirement depends on the tile dimensions. Reading back along the smaller side results in a smaller internal memory requirement and therefore will be preferred. From this point on it is assumed that $\frac{n_1}{k_1} \le \frac{n_2}{k_2}$, and reading back is performed along the vertical direction. After writing is completed, the last S pixels are read from the memory.

Since Reversible Color Transform (RCT) [19] requires R, G, and B components, it should be applied before the image data is buffered for P<3. However for P≥3, RCT can be also applied after tiling.

Since for each write operation a read-back is performed, this scheme requires a storage-processor link with a bandwidth which is twice the bandwidth of the image input stream.

Symmetric extension may be applied at the boundaries of the tiles, since it eliminates the high frequency content inherent in boundary treatments such as zero padding, or circular convolutions and makes perfect reconstruction possible for symmetric filters. In addition to the architectural complexities introduced to the system (which are discussed in Section 3.5), performing 2-D DWT on tiles also introduces blocking artifacts, which are present as similar in traditional JPEG images [55] at the boundaries whenever a lossless filtering or recovering procedure is not preferred. Especially when sub-bands are discarded and further quantization is applied, these artifacts may result in considerable perceptual quality degradations [54] [55]. This is due to the fact that the boundary coefficients are computed with false values obtained by symmetric extension. However, if instead of reflection pixels the genuine boundary pixels are used such degradation will never occur. Moreover, the smaller tile sizes can be chosen in order to reduce the internal memory. To avoid such degradation, the overlap reading method can be applied.

Without the application of symmetric extension the values at the boundaries can be obtained by reading extra area (extension area) around each tile which is already available in off-chip storage. This method also reduces the logic in the filters that is due to symmetric extension (see Section 3.5).

Let a and b be the extension needed at the beginning and ending boundaries of a row respectively., i.e. at least one of $g_0^{(j)}$ and $h_0^{(j)}$ depends on $h_{-a}^{(j-1)}$ and at least one of $g_{r/2-1}^{(j)}$ or $h_{r/2-1}^{(j)}$ depends on $h_{r+b-1}^{(j-1)}$ for a row length of r. Figure 3.13 shows the one dimensional case of a filtering operation. For the sake of illustration let us assume r=16, a=2 and b=1, which is the case for a lifting implementation of the 5/3 filter [15]. For the odd-length symmetric filters $a+b=l-2$ where l is the length of the longer filter.

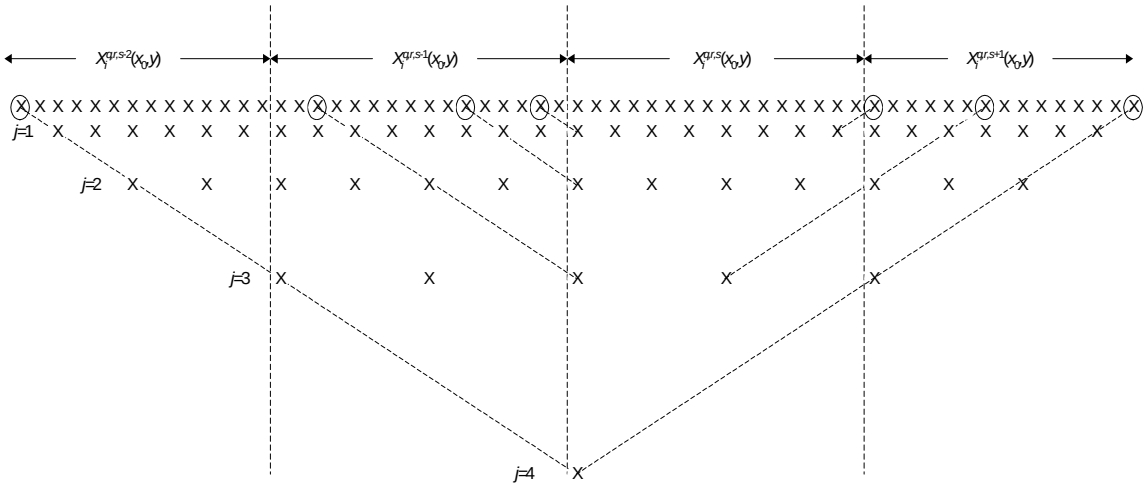


Figure 3.13 Overlap reading for one-dimensional transform for r=16, a=2 and b=1

The overall number of extra reads performed due to overlap reading for a 1-D J level transformation is :

$$e = e_1 + e_2 = (l-2)\cdot\sum_{j=1}^{J} 2^{j-1} \tag{3.21}$$

Figure 3.14 shows the overlap reading for 2-D wavelet transform.

Reading extra pixels from storage results in a bandwidth expansion between the storage and the processing unit. Note that the required bandwidth depends on the location of the tile which is being processed since overlap reading is not applied for the boundaries coinciding with those of the entire frame. However the maximum expansion factor in number of reads performed can be calculated as:

$$R_{\text{ext}} = \left(1+\frac{ek_1}{n_1}\right)\cdot\left(1+\frac{ek_2}{n_2}\right) \tag{3.22}$$

and the overall bandwidth expansion factor is

$$\frac{1+R_{\text{ext}}}{2} \tag{3.23}$$

For example, if we would like to apply 3 levels of 5/3 transform to tiles of size 256×256, the expansion factor is only 1.09. If we would like to apply 5 levels of transform, this time the expansion factor becomes 1.43, which may be intolerable for our application. Furthermore, if we would like to apply 9/7 transform of 5 levels to tiles of size 64×64 (we may be concerned about the internal storage size), the expansion factor turns out to be 10.14, which results in a gigantic read/write requirement.

With the proposed method blocking artifacts are eliminated. However, for smaller tile sizes, larger levels of decomposition, or longer filter kernels, the bandwidth increases rapidly and the method loses its practicality.

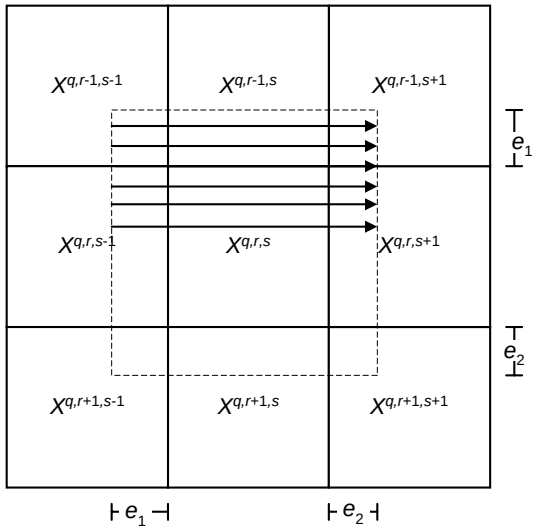


Figure 3.14 Overlap reading for two dimensional transform

3.7 Cascade Filter Structure

Various implementations have been summarized in Chapter 2. An FPGA implementation of the 2-D wavelet transform processing large image frames may exhibit a large routing network due to access to distributed RAM blocks which are usually distributed over the entire FPGA IC. A centralized approach to RAM access, by means of a single control unit that manages all levels sub-band generation, or that has complex scheduling algorithms or intermediate sub-band multiplexing may also introduce excessive logic to the system. Moreover, lifting structures which ensures considerable memory reductions and signal boundary handling which is inevitable for most applications may pose a problem in terms of circuit complexity in such architectures [39][57]. A more straightforward approach to the problem, which uses cascaded 2-D filtering blocks each controlling its own memory space in parallel is a more appropriate choice for resolution levels up to a certain number. A similar approach is discussed in [45] as a pipelined approach. However the discussed architecture does not use RAM blocks but shift registers. In our application, though, memory usage and independent memory space for each level is essential.

Figure 3.15 shows the proposed cascaded DWT structure. *j*th resolution level is computed by the 2-D DWT module at the *j*th stage. Except for their directions of filtering, both the horizontal and the vertical filtering perform the same type of computation. In hardware implementations, however, the vertical filter requires several whole rows to be stored in memory during processing. Vertical filtering thus, consume much more area than the horizontal filtering. Besides, filters of vertical and horizontal directions have some structural differences in that the vertical filter utilizes internal RAM storage whereas the horizontal filter uses only shift registers.

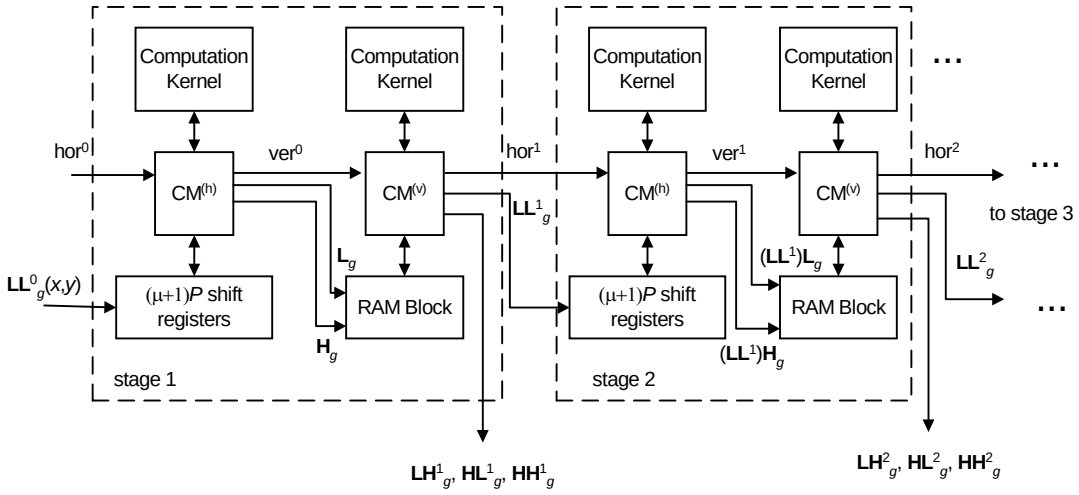


Figure 3.15 Proposed cascaded filter architecture. *j*th resolution level is computed by the 2-D DWT module at the *j*th stage. Row beginning/endings are asserted by the signal $\mathtt{hor}^j$. Tile beginning/endings are asserted by signal $\mathtt{ver}^j$.

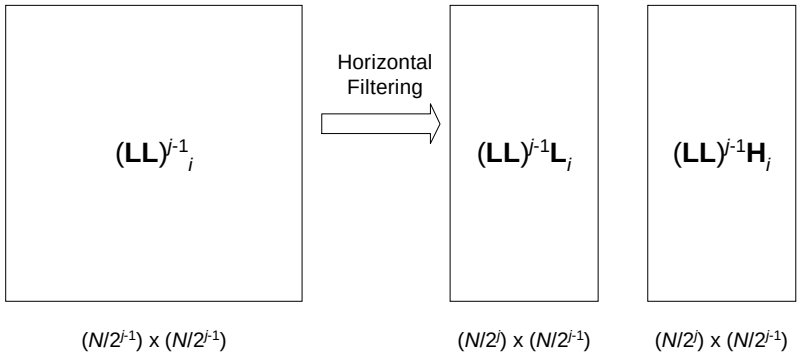


Figure 3.16 Horizontal filtering and decomposition of LL subband

3.7.1 Horizontal Filter

This block is the part which handles the filtering along the horizontal direction. The term "horizontal direction" implies that filtering is along the 'rows'. Since what the filter receives as 'rows' may be along any direction, from the perspective of the

whole image, this module may actually be processing "vertical" filtering. (see Section 3.6)

The horizontal filter operates on the pixels in one row of the input sub-band. As shown in Figure 3.16, horizontal filter at level j divides the input sub-band $(\mathbf{LL})^{j-1}{}_i$ which is of size $(N/2^{j-1}) \times (N/2^{j-1})$ into $(\mathbf{LL})^{j-1}\mathbf{L}_i$ and $(\mathbf{LL})^{j-1}\mathbf{H}_i$ frames of size $(N/2^j) \times (N/2^{j-1})$, where

$$N = \frac{n_1}{k_1} \tag{3.24}$$

For the sake of simplicity, it is assumed that the image and tile sizes are chosen such that $(n_1/k_1) < (n_2/k_2)$.

To comply with the JPEG 2000 standard [19] symmetric extension may be applied at the row boundaries.

Row-based acquisition of input frames does not pose a problem in horizontal filtering of the incoming rows, since they can be filtered as they are received. Each row can be filtered independently of adjacent rows with boundary manipulation. This requires a small time gap between two consecutive rows, however for resolution levels other than the first level this gap is inherent. For the 0th level –i.e. for the input time variant boundary filtering can be used (see Section 3.5).

The row length is programmable. Input row length to a horizontal filter at any level can be of any even number greater than or equal to 4 (This requires that the first level input length should be multiple of 2^J greater than or equal to 2^{J+1}). As it was mentioned earlier, row length determines the memory requirement for the system. Therefore it is upper bounded with the memory available. Since the row length is not a fixed value, horizontal filter needs to be aware of the row beginning/endings. This is done by the signal $\mathtt{hor}^j$. When this signal is high the row transmission occurs. This method provides a design simplification since the horizontal filter in each level is identical, and has no counters.

3.7.2 Vertical Filter

This block receives the low-pass and high-pass data sequences and filters them vertically. Like the horizontal filter, this module also uses lifting steps to filter the data.

The vertical filter operates on the rows of the input block. Horizontal filter at level j divides the input blocks $(\mathbf{LL})^{j-1}\mathbf{L}_i$ and $(\mathbf{LL})^{j-1}\mathbf{H}_i$ which are of size $(N/2^j)$ x $(N/2^{j-1})$ into sub-bands $(\mathbf{LL})^j_i$, $(\mathbf{LH})^j_i$, $(\mathbf{HL})^j_i$ and $(\mathbf{HH})^j_i$ which are of size $(N/2^j)$ x $(N/2^j)$ as shown in Figure 3.17.

In vertical processing, row-based acquisition requires whole rows to be treated as a single pixel. This imposes a memory requirement which is proportional to the row length and this memory requirement is therefore much larger than that of horizontal filtering.

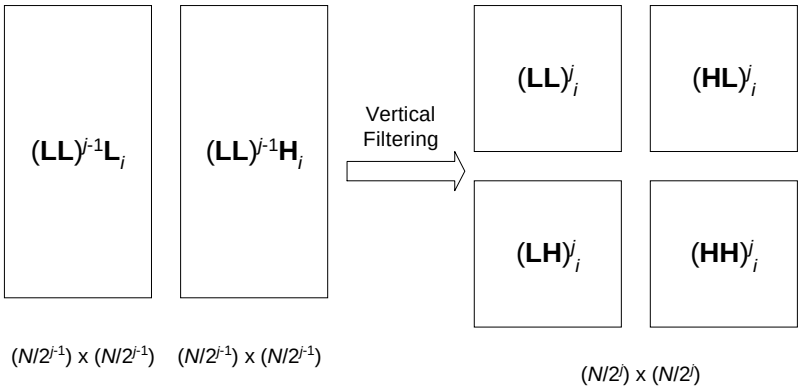
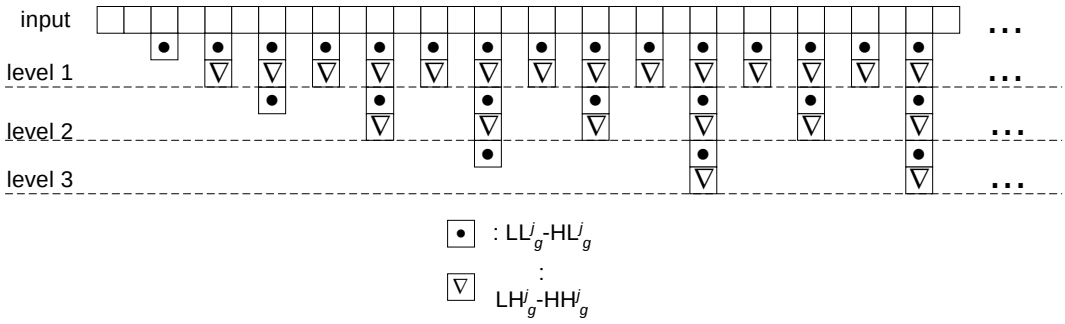
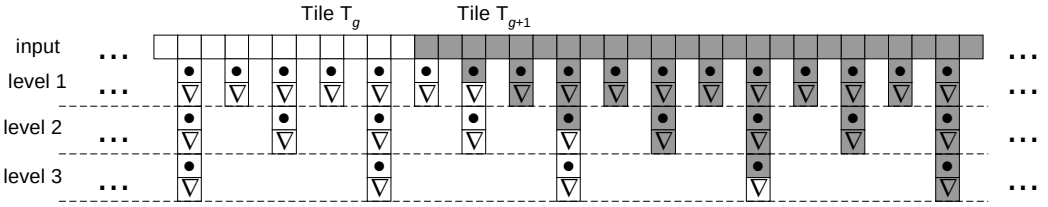


Figure 3.17 Vertical filtering and decomposition of (LL)H and (LL)L subbands.



(a)



(b)

Figure 3.18 Output timing diagrams of three cascaded 2-D filters. (a) for g=0, (b) for $0<g<Z$

In Figure 3.18, output timing diagrams of 3 cascaded 2-D filters are illustrated for the case when the horizontal and vertical filters have the architecture as explained in Section 3.4. Figure 3.18a illustrates the tile g=0 and 5.18b illustrates $0<g<Z$.

For the sake of illustration, the integer 5/3 filter, which provides a better demonstration of the case, is shown. Each box represents a row of data from sub-bands belonging to the corresponding resolution level. A row from resolution level j contains $N/2^j$ pixels and each level contains $N/2^j$ rows.

Figure 3.18a shows the case for the first tile. Each level receives its input from the previous level -say *input level*- and releases its data after a delay of d=2 rows (see Section 3.4).

In Figure 3.18b the joint of two consecutive tiles is illustrated. It is apparent that the DWT stages do not finish with a tile at the same time. As long as the input rows keep arriving at a stage, the DWT module at that stage maintains its outputting scheme regardless of the tile beginning/endings. Symmetric extensions are performed by location variant filters which use the multiplexing among the buffers explained in Section 3.5. Hence the buffers are fully utilized and there is no need for gaps between the adjacent tiles. The shading of each box in Figure 3.18b denotes to which tile the rows belong.

Like the row length, the column length is also programmable. Input column length to a level of decomposition may be any number greater than or equal to 4. Since the columns are not constrained, unlike the rows, they can be infinitely long. Hence, the architecture is suitable for space imaging applications employing linear sensors which scan the earth's surface generating an unbounded data in vertical axis.

Vertical filter needs to be aware of the tile beginning/endings in order to handle symmetric extension. This is done by the signal $\mathtt{ver}^j$. When this signal is high, row transmission occurs. Low to high transition indicates a tile beginning whereas high to low transition indicates a tile ending.

3.7.3 Memory Requirements

The system discussed in detail in sections 3.4-6, 3.7.1-2 uses row-based processing in which a minimum amount of data to compute the output is stored in buffers. Figure 3.19 shows the part of the image which is inside the 2-D DWT processor.

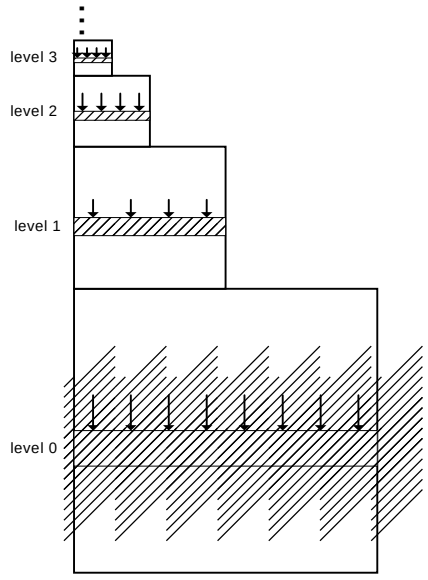


Figure 3.19 Part of the image allocated in filter memories.

Since the memory required to compute a frame $\overset{\pi}{F}$ is the same for every π , ($0 \leq \pi < P-1$), one can deduce that the memory required to compute a single frame is :

$$\mathrm{Mem}\{\overset{\pi}{F}\} = \frac{1}{P}\mathrm{Mem}\{\mathbf{F}\} \tag{3.25}$$

where **F** is the concatenated frame and P is the number of parallel processing units. Therefore only computation of concatenation frames are taken into consideration.

The memory required by the horizontal filters to compute $(\mathbf{LL})^{j-1}\mathbf{L}_g$ and $(\mathbf{LL})^{j-1}\mathbf{H}_g$ in terms of pixels is independent of the level and is:

$$m_j^{(h)} = (\mu + 1)\cdot P \tag{3.26}$$

and the memory required by the vertical filters to compute $(\mathbf{LL})^j_g$, $(\mathbf{LH})^j_g$, $(\mathbf{HL})^j_g$ and $(\mathbf{HH})^j_g$ in terms of pixels is:

$$m_j^{(v)} = \left(\frac{n_1}{k_1}\right) \cdot (\mu + 1) \cdot P \cdot 2^{1-j} \tag{3.27}$$

where μ is the total memory required to implement the lifting steps defined in (3.12), and $\frac{n_1}{k_1}$ is the smaller side dimension of the tiles. The term (μ+1) is due to the fact that one more delay element is used in order to split the signals.

The total memory requirement can be found by summation of all $m_j^{(h)}$ and $m_j^{(v)}$ for all the levels 1 through J :

$$\begin{aligned} m &= \sum_{j=1}^{J} \left(m_j^{(v)} + m_j^{(h)}\right) \\ &= \sum_{j=1}^{J} \left(\frac{n_1}{k_1} 2^{1-j} + 1\right) \cdot (\mu + 1) \cdot P \end{aligned} \tag{3.28}$$

3.7.4 Output Bandwidth Considerations

The link between the 2-D DWT processor and the forthcoming units such as memory blocks for coefficient storage or CPUs handling entropy coding may not be able to accommodate the high-bandwidth output generated by the DWT processor. The scheme in Figure 3.18 results in a "burst-full" output generation, i.e., all stages are in silent mode and in transmitting mode during the same time intervals. Therefore, although the amount of data received and produced is the same (dynamic range expansion is not taken in to account), the output bandwidth turns out to be larger than that of input and may exceed the output link capabilities. For some cases, the high output bandwidth may require large buffers in order to compensate for the stall periods of the output link.

The maximum bandwidth requirement within a row transmission, i.e. at the instance where stages are transmitting, for the scheme discussed in previous section can be given as :

$$B_{\text{out}} = \left(\frac{1}{2^J} + \sum_{j=1}^{J} \frac{3}{2^j} \right) \cdot B_{\text{in}} \tag{3.29}$$

which converges to $3B_{\text{in}}$ for large J. We can solve this problem and eliminate the triple bandwidth requirement with some modifications to the cascade filter structure. Due to the sub-sampling by two present in sub-band decomposition, each resolution level can be interspersed between the data of the parent resolution level. The sub-bands can be scheduled so as to exploit this property. With a modification in processors of each stage accept the last one a burst-free transmission can be achieved.

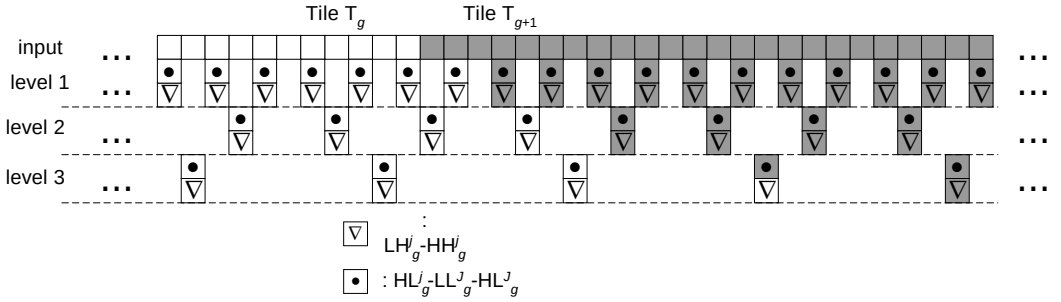


Figure 3.20 “Burst-free” output timing diagram for the modified structure

Note that in the natural output scheme illustrated in Figure 3.18 sub-band rows $(\mathbf{HH})^j_g\{x_0,y\}$ and $(\mathbf{LH})^j_g\{x_0,y\}$ are computed at the same time with $(\mathbf{LL})^{j-1}_g\{2x_0+d+2,y\}$ and $(\mathbf{HL})^j_g\{x_0,y\}$ is computed at the same time with $(\mathbf{LL})^{j-1}_g\{2x_0+d,y\}$. If $(\mathbf{HH})^j_g$, $(\mathbf{LH})^j_g$ and $(\mathbf{HL})^j_g$ (excluding the ones in the last level) are scheduled so that these sub-bands are released as $(\mathbf{LL})^{j-1}_g\{2x_0+d+1,y\}$ is computed, we end up with an output transmission which is burst free as shown in Figure 3.20. For the lifting structures which have $N_p=N_u$ (see Section 3.4) $(\mathbf{HH})^j_g$ and $(\mathbf{LH})^j_g$ can be supplied earlier than their natural scheduling time from the memory, however $(\mathbf{HL})^j_g$ requires an additional memory of a unit row size. The modified structure will have the memory requirement of :

$$m_j'^{(v)} = \begin{cases} \left(\frac{n_1}{k_1}\right)\cdot\left(\mu+\frac{3}{2}\right)\cdot P\cdot 2^{1-j} & 0\le j<J, \\ m_j^{(v)} & j=J \end{cases} \tag{3.30}$$

and,

$$m' = J\cdot(\mu+1)\cdot P + \left(\frac{n_1}{k_1}\right)\cdot(\mu+1)\cdot P\cdot 2^{1-J} + \left(\frac{n_1}{k_1}\right)\cdot\left(\mu+\frac{3}{2}\right)\cdot P\cdot\sum_{j=1}^{J-1}2^{1-j} \tag{3.31}$$

The new maximum bandwidth over a row is :

$$B'_{\text{out}} = \frac{3}{2}\cdot B_{\text{in}} \tag{3.32}$$

3.8 Precision and Channel Constraints

As it is explained in Section 3.2 input data to the transformer is *B*- bit signed having zero offset. Unfolded structure makes it possible to assign different precision for the computation of each sub-band. Each sub-band $\beta^{(j)} \in \{\mathbf{LL}^{(j)}, \mathbf{LH}^{(j)}, \mathbf{HL}^{(j)}, \mathbf{HH}^{(j)}\}$, is computed in $B+\Gamma_{\beta^{(j)}}$ bit precision, where

$$\Gamma_{\beta^{(j)}} = \mathrm{X}_{\beta^{(j)}} + G_{\beta^{(j)}} \tag{3.33}$$

$\mathrm{X}_{\beta^{(j)}}$ accounts for the resultant nominal gain of the filtering operations involved in computation of $\beta^{(j)}$. For the 5/3 filter the value of $\mathrm{X}_{\beta^{(j)}}$ is :

$$\mathrm{X}_{\mathrm{LL}^{(j)}} = 0, \quad \mathrm{X}_{\mathrm{LH}^{(j)}} = \mathrm{X}_{\mathrm{HL}^{(j)}} = 1, \quad \text{and } \mathrm{X}_{\mathrm{HH}^{(j)}} = 2, \qquad \forall j \tag{3.34}$$

$G_{\beta^{(j)}}$ is the number of extra guard bits in order to prevent any computation to fall beyond the nominal range bounds. A typical value for *G* is 1, or more conservatively *G*=2 [10].

Since for achieving high rates of compression quantization is already applied to coefficients, some bits can be omitted prior to the coding process (resulting a uniform quantization in coefficients) for the following reasons:

i. The channel bandwidth between DWT processor and entropy coder or the coefficient storage may not be high enough for the throughput requirements,

ii. Large coefficient storage may not be feasible for the system configuration,

iii. Rearrangement of coefficients in order to store $B+\Gamma$ bit data compactly in multiple-of-B bit storage unit may require excessive amount of logic or buffers in reconfigurable logic.

Applying uniform quantization by discarding q_β LSB's of the coefficients is equivalent to setting the $K_\beta^{\max}$ of the entropy coder [19] to :

$$K_\beta^{\max} = B + \Gamma_\beta - q_\beta - 1 \tag{3.35}$$

Note that in the above discussion it is assumed that the input data is not color transformed. For the case of color transformation, input precision for particular components should be taken as $B+1$ as explained in Section 3.3.

High frequency sub-bands HH, LH and HH usually consist of very small coefficients centered around zero. Therefore these sub-bands can be transmitted in a number of bits smaller than $B+\Gamma$ by clamping rather than quantizing, or both can be applied. With this method risking some large coefficients to fall beyond the range, we can get rid off the necessary quantization. The clamped and quantized coefficients of sub-band β is expressed by :

$$\hat{\beta} = \begin{cases} \left\lfloor \dfrac{\beta}{2^{q_\beta}} \right\rfloor & 2^{\theta_\beta + q_\beta - 1} \le \beta \le 2^{\theta_\beta + q_\beta - 1} - 1, \\ 2^{\theta_\beta - 1} - 1 & \beta > 2^{\theta_\beta + q_\beta - 1} - 1, \\ -2^{\theta_\beta - 1} & \beta < -2^{\theta_\beta + q_\beta - 1} \end{cases} \tag{3.36}$$

where θ_β is the new precision for transmission.

CHAPTER 4

IMPLEMENTATION AND EXPERIMENTAL RESULTS

This chapter is organized as follows: Section 4.1 presents the implementation of the architecture that is explained in detail in Section 3. In Section 4.2 comparisons of the architecture with the present architectures in literature and the comparisons of the JPEG 2000 compression achievements are given. Section 4.3 presents the simulation results obtained.

4.1 FPGA Implementation

The hardware is part of a JPEG 2000 compression system, designed as a payload for a Low Earth Orbit (LEO) micro-satellite, which will be launched in September 2003. Figure 4.1 is a photograph of GEZGİN [23], the payload for JPEG 2000 image compression. The design is implemented on a XILINX XCV300EPQ240-6 IC (The large black package populated on the top left portion of the circuit board in Figure 4.1) [47].

Figure 4.1 Photograph of GEZGİN, the image compression system which is designed as a payload for a LEO micro-satellite, BILSAT-1. (The photograph is provided with the courtesy of TÜBİTAK-BİLTEN).

4.1.1 Specifications

The hardware receives multi-spectral image data from four cameras (M=4) simultaneously. The number of parallel DWT processors (see Figure 3.1) is 2 (P=2). Since P<3, RCT is applied prior to the tiling step. For this application two different tile sizes of 256 and 128 ($n_1/k_1 = n_2/k_2$ = 128, 256) are used. Although the tile size can be run-time programmable, since commanding FPGA would result in a more complex circuit board topology, it is preferred to be fixed for this application The image streams are received in quadrants as explained in Section 3.6. The hardware employs 3 levels of sub-band decomposition (J = 3) in which 5/3 integer filtering [19] is used. In order to prevent burst output generation and the congestion at the output link, a burst-free scheme is preferred. The local storage size required for the tiling operation is S=2 MB.

The throughput requirement of the hardware is 80Mbps (Four cameras streaming out at 20 Mbps each). The two DWT processors are operated at 20 MHz, however interface modules are operated at 80 MHz in order to increase the RAM excess

bandwidth and to sample the input data properly. The filter is capable of handling tiles up to 512×512 in size, but this is not preferred due to the increased requirement for local storage and transform latency. The DWT processor block can be operated at frequencies up to 40MHz. This results in a throughput capability of 160Mbps. When the hardware operates at 80Mbit/s using 256×256 size tiles, the latency introduced is 0.105 sec (compared to a total transmission time of 2048×2048×8/(20Mbit/s) = 1.678 sec).

4.1.2 FPGA Operation Environment

Figure 4.2 shows the block diagram of the image compression system. Transformed data is transmitted through HPI link and stored in coefficient storage. Coefficients are entropy coded by DSP which implements MQ coder of [19]. Operation mode of entropy coder is "run-mode" with code-block size of 32 for tile size 256 and 16 for tile size 128.

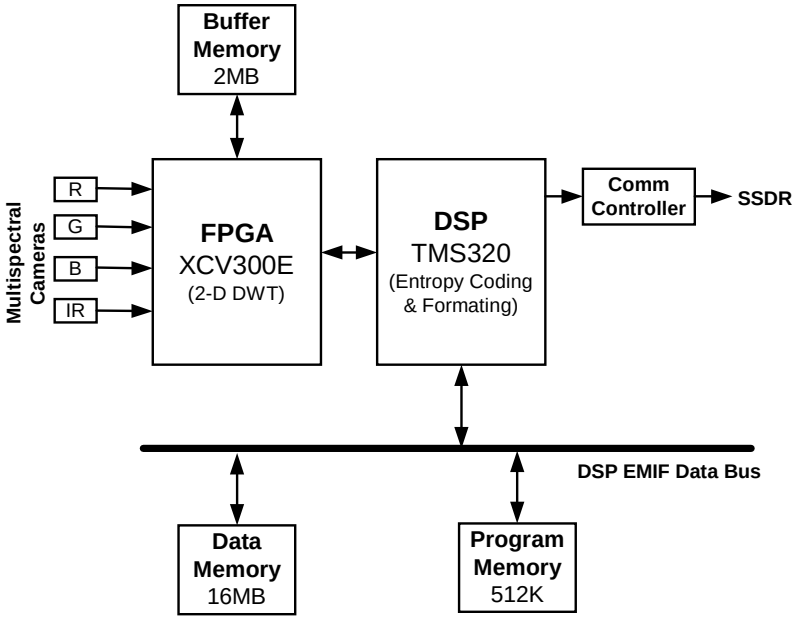


Figure 4.2 Block diagram of the image compression system.

The XILINX FPGA communicates with LVDS interface chips from which it receives the multi-spectral image data, SRAM buffers which are used to partition the input image, DSP which implements the entropy coding and data formatting, configuration memory from which the FPGA downloads its configuration data.

4.1.3 Design

This section introduces the design environment and synthesized chip.

4.1.3.1.Design Environment

The Design is implemented on a XILINX XCV300EPQ240-6 IC [47]. Before the GEZGİN implementation, the hardware was implemented and verified on CELOXICA RC1000a prototyping card [58], which populates a XCV2000E FPGA, 8MB SRAM and a PCI host interface. For the synthesis and front-end design SYNOPSYS FPGA Express 2000 v.3.5 is used. Design entry is in Verilog HDL, a hardware description language. Place & route is done on Xilinx Flow Engine v3.3.08i. Simulations and tests are done in MATLAB and GEZGİN Test and Decoder Suite v1.0 [67] provided with the courtesy of TÜBİTAK-BİLTEN.

4.1.3.2.Synthesized Chip

The top level design has 9 structural modules. These are : trig and reset module, lvds interface, sequencer (buffering and tiling module), frequency adaptor, reversible color transformer (RCT module), 2-D DWT module, HPI module, HPI multiplexor, SRAM interface. All these modules are briefly explained in Appendix A.

Figure 4.3 shows the schematic of the synthesized computation kernel explained in Section 3.4 and 3.8, which implements four 10-bit adders and one 2's complementor.

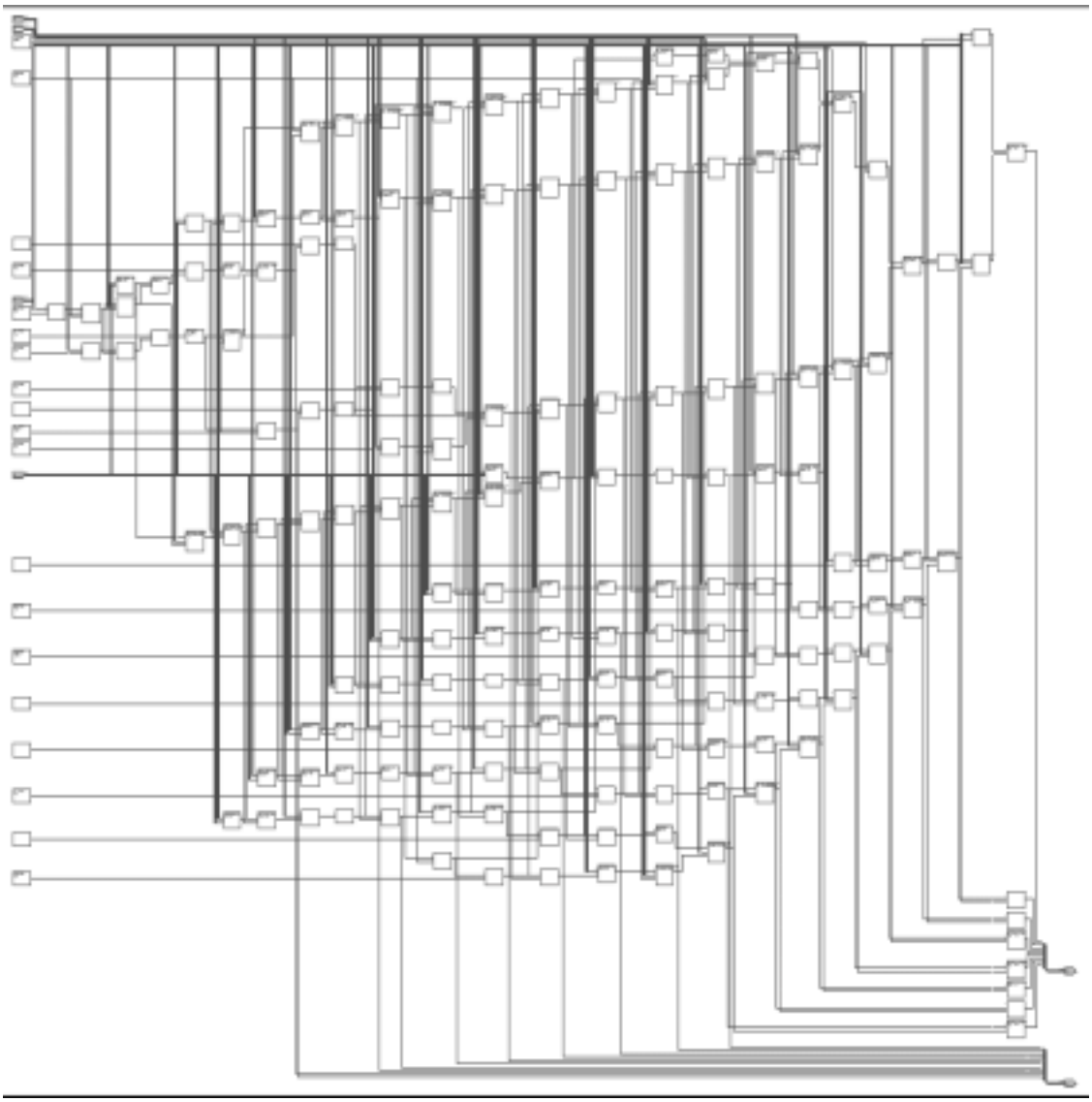


Figure 4.3 The schematic of the synthesized computation kernel, which implements four 10-bit adders and one 2's complementor.

4.1.4 Overall Architecture

Figure 4.4 shows the floorplan of the design. The figure is provided to give an idea about the logic placement of figures. Area consuming modules are contoured and numbered. The hierarchy is not flattened during synthesis[1].

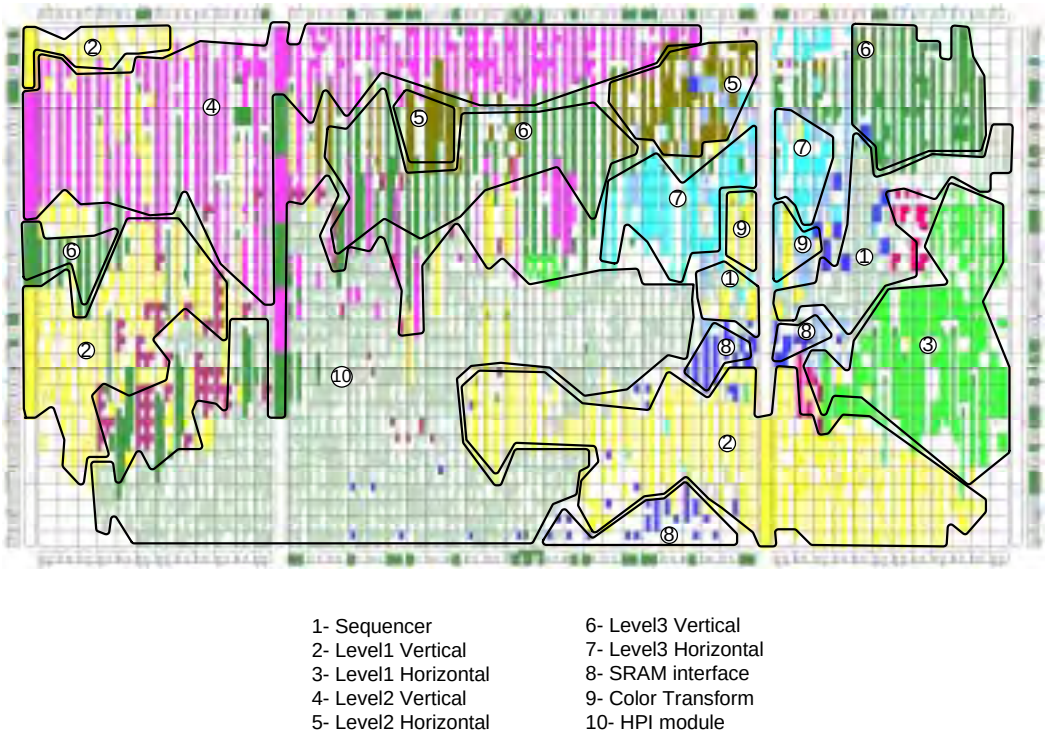


Figure 4.4 The floorplan of the design. Area consuming modules are contoured and numbered

Table 4.1 shows the resources used in the DWT module and resources available in the XCV300EPQ240 chip. Equivalent gate count for the design is 292,447.Table 4.2 shows the detailed resource usage of the hierarchical modules. Throughout the

[1] The motivation for the non-flattened hierarchy has actually been a non-systematic failure of the flattened hardware encountered during GEZGİN tests, which may possibly be due to the skew at internal clock boundaries. Although the flattened hierarchy results in a slight reduction in resource usage and a slight increase in minimum clock periods, it could not be verified, and therefore is not reported in this work.

design maximum fanout constrained by 20. Table 4.3 shows the BRAM modules required in each level for various implementations for $n_1/n_2=n_2/k_2=N$ and P=2. Shaded column corresponds to GEZGİN's case. In Appendix B cells of the device is explained.

Table 4.1 Resources used in DWT module and resources available in XCV300EPQ240 chip

Resource	Number Used	Out of
Slices	*2,778*	*3,072*
Bonded IO Buffers	*79*	*158*
GCLKIOBs	*2*	*4*
GCLKs	*4*	*4*
DLLs	*2*	*8*
Slice FFs	*1,784*	*6,144*
DWT	*679*	
Rest	*1,105*	
LUTs (FGs)	*4,473*	*6,144*
DWT	*2,963*	
Rest	*1,510*	
Block RAMs	*14*	*32*
DWT	*14*	
Rest	*-*	

Table 4.2 Detailed resource usage of the hierarchical modules

Hierarchy			FG	CY	DFF	BRAM
sequencer			37	46	183	-
freq_adapt			16	-	28	-
lvds_inf			57	20	118	-
reset_module			13	-	8	-
ram_inf			42	-	104	-
rct			53	34	33	-
hpi_mux			99	-	75	-
hpi_mdl			1191	-	555	-
DWT			-	-	-	-
	Lev3		-	-	-	-
		Hor	100	-	109	-
		kernel1	44	49	-	-
		kernel2	44	49	-	-
		total	188	98	109	-
		Vertical	569	7	114	-
		kernel1	40	47	-	-
		kernel2	40	47	-	-
		kernel3	40	47	-	-
		kernel4	40	47	-	-
		ram_module	-	-	-	4
		total	729	195	114	4
		total	917	293	223	4
	Lev2		-	-	-	-
		Hor	100	-	109	-
		kernel1	44	49	-	-
		kernel2	44	49	-	-
		total	188	98	109	-
		Vertical	612	-	72	-
		kernel1	40	47	-	-
		kernel2	40	47	-	-
		kernel3	40	47	-	-
		kernel4	40	47	-	-
		ram_module	-	-	-	5
		total	772	188	72	5
		total	960	286	181	5
	Lev1		-	-	-	-
		Hor	217	-	203	-
		kernel1	44	49	-	-
		kernel2	44	49	-	-
		total	305	98	203	-
		Vertical	621	-	72	-
		kernel1	40	47	-	-
		kernel2	40	47	-	-
		kernel3	40	47	-	-
		kernel4	40	47	-	-
		ram_module	-	-	-	5
		total	781	188	72	5
		total	1086	286	275	5
	total		2963	865	679	14
total			4472	1064	1784	14

Table 4.3 BRAM modules required in each level for various implementations for $n_1/n_2=n_2/k_2=N$ and $P=2$

Module	$J=2$			$J=3$		
	$N=128$	$N=256$	$N=512$	$N=128$	$N=256$	$N=512$
Level 1	*5*	*5*	*10*	*5*	*5*	*10*
Level 1	*4*	*4*	*4*	*5*	*5*	*5*
Level 1	*-*	*-*	*-*	*4*	*4*	*4*
Total	*9*	*9*	*14*	*14*	*14*	*19*

4.1.5 Power, Timing and Test Subjecting

The hardware uses three clock signals with periods of 12.50 ns (CLK80), 25.00 ns (CLK40), and 50.00 ns (CLK20) each having 50% duty cycles. These clocks are generated form the internal DLL circuitry provided on the IC. Table 4.4 shows the clock groups and which hierarchical modules contain these groups. Note that the Trig-and-Reset Module does not contain any of these clock groups since it is clocked by the external system clock of 20 MHz. Maximum delays from one clock group to itself and to the others, and the number of logic levels (routing and FG) contained by the critical paths are tabulated in Table 4.5.

Table 4.4 Clock groups

Clock Group	Modules
CLK20	*HPI module (Partly), DWT module, Freq. Adaptor (Partly)*
CLK40	*HPI module (Partly)*
CLK80	*LVDS interface (Partly), RCT, Sequencer, SRAM interface, Freq. Adaptor (Partly)*

Table 4.5 Maximum path delays for clock groups

Path	Constrained Value (ns)	Actual (ns)	# Logic Levels
CLK20 to CLK20	*50.000*	*44.073*	*21*
CLK20 to CLK40	*25.000*	*14.701*	*3*
CLK20 to CLK80	*10.000*	*7.376*	*4*
CLK40 to CLK20	-	-	-
CLK40 to CLK40	*25.000*	*22.623*	*6*
CLK40 to CLK80	-	-	-
CLK80 to CLK20	*18.000*	*11.173*	*5*
CLK80 to CLK40	*25.000*	*8.368*	*6*
CLK80 to CLK80	*10.000*	*10.075*	*7*

Power consumption of the device is estimated by using Power Estimator provided at [59]. The parameters to the estimator is supplied larger than original values in order to allow for a safety margin. Table 4.6 shows the estimated internal power consumption in CLB logic, Block RAMs and DLLs, output power consumption in output pins and the device quiescent power during processing time (i.e, while the camera data reception occurs). The design and physical layout of the GEZGİN module does not provide a means for measuring the power consumption of individual ICs. The module is operated at 28 V and has a supply current of ~230 mA and a power consumption of ~6.44 W during full processing. The power consumption of the DWT hardware is estimated to be about 400 mW which is only 6% of the total system power. This estimation is also supported with the observed current characteristics of the module during FPGA run-time.

Table 4.6 Estimated power consumption

Device Quiescent Power	*36 mW*
CLB Logic Power	
CLK20 Logic	*55 mW*
CLK40 Logic	*38 mW*
CLK80 Logic	*184 mW*
Total CLB Logic Power	*278 mW*
Block SelectRAM Power	*14 mW*
Clock DLL Power	
DLL 1 @20MHz	*6 mW*
DLL 2 @40MHz	*10 mW*
Total Clock DLL Power	*16 mW*
Total Estimated Internal Power @1.8V	344 mW
Input/Output Power	*56 mW*
SRAM Interface	*40 mW*
LVDS Interface	*0 mW*
HPI	*16 mW*
Total Input/Output Power	*56 mW*
Total Estimated External Power @3.3V	56 mW
Total Estimated Power	**400 mW**

The GEZGİN module was subjected to exhaustive tests such as -20C to 50C temperature cycling, vibration and continuous operation. It passed all tests.

4.2 Comparisons

4.2.1 Resource Used

Table 4.7 shows the resource used by 2-D DWT processor implemented for GEZGİN [23] and various 2-D DWT architectures. For this comparison each architecture is assumed to realize the 2-D DWT with 3 levels of sub-band decomposition and to use 5/3 filters. Tile size is assumed to be 256×256. Asymptotic internal storage requirement is given for large J. For J=3 the internal storage requirement is 1344 pixels (for "burst-full" case) and 1536 pixels (for "burst-free" case).

Table 4.7 Resource used by 2-D DWT processor implemented for GEZGİN and various 2-D DWT architectures.

Architecture	Storage Size (pixels) (for large J)	Adders (for J=3)	Multipliers (for J=3)	Computation Time (ccs)
Direct[32]	65536	*5*	*3*	$4N^2$
SIMD[33]	131072	*256-512*	*256-512*	$4JL$
Parallel 1[37]	≈ 2944	*20*	*12*	$\approx N^2$
Parallel 2[37]	≈ 2816	*24*	*18*	$\approx N^2$
Masud[39]	≈ 2560	*16*	*11*	$3/2\ N^2$
Non-Separable[33]	2560	*32*	*18*	N^2
Systolic-Parallel[32]	2560	*18*	*12*	N^2+N
Row-parallel [40]	$1536J$	*1024(FA)*	-	$\approx N^2+N$
Lattice [41]	2590	*16*	*12*	$N^2/2$
Level-by-Level[42]	18944	*16*	*12*	$N^2/2$-0.67 N^2
Semi-recursive[44]	≈65536	*12*	*12*	$4N^2/3$
Quadri-Filter Folded [45]	2560	*13*	*13*	$2/3\ N^2$
Quadri-Filter Piple-lined[45]	2560	*39*	*39*	$N^2/2$
Our Design	1536-1792	*36*	*9*	$N^2/2$

4.2.2 JPEG 2000 Achievement

For this discussion the Peak Signal-to-Noise Ration (PSNR) is computed by the formula:

$$\mathrm{PSNR} = 20 \cdot \log_{10} \frac{2^B - 1}{\sqrt{\mathrm{MSE}}} \tag{4.1}$$

where B is the bit-depth of the samples, which is equal to 8 in our application, and MSE (mean square error) is calculated by:

$$\mathrm{MSE} = \frac{1}{n_1 n_2} \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \left[x(i,j) - \hat{x}(i,j) \right]^2 \tag{4.2}$$

for monochrome images and

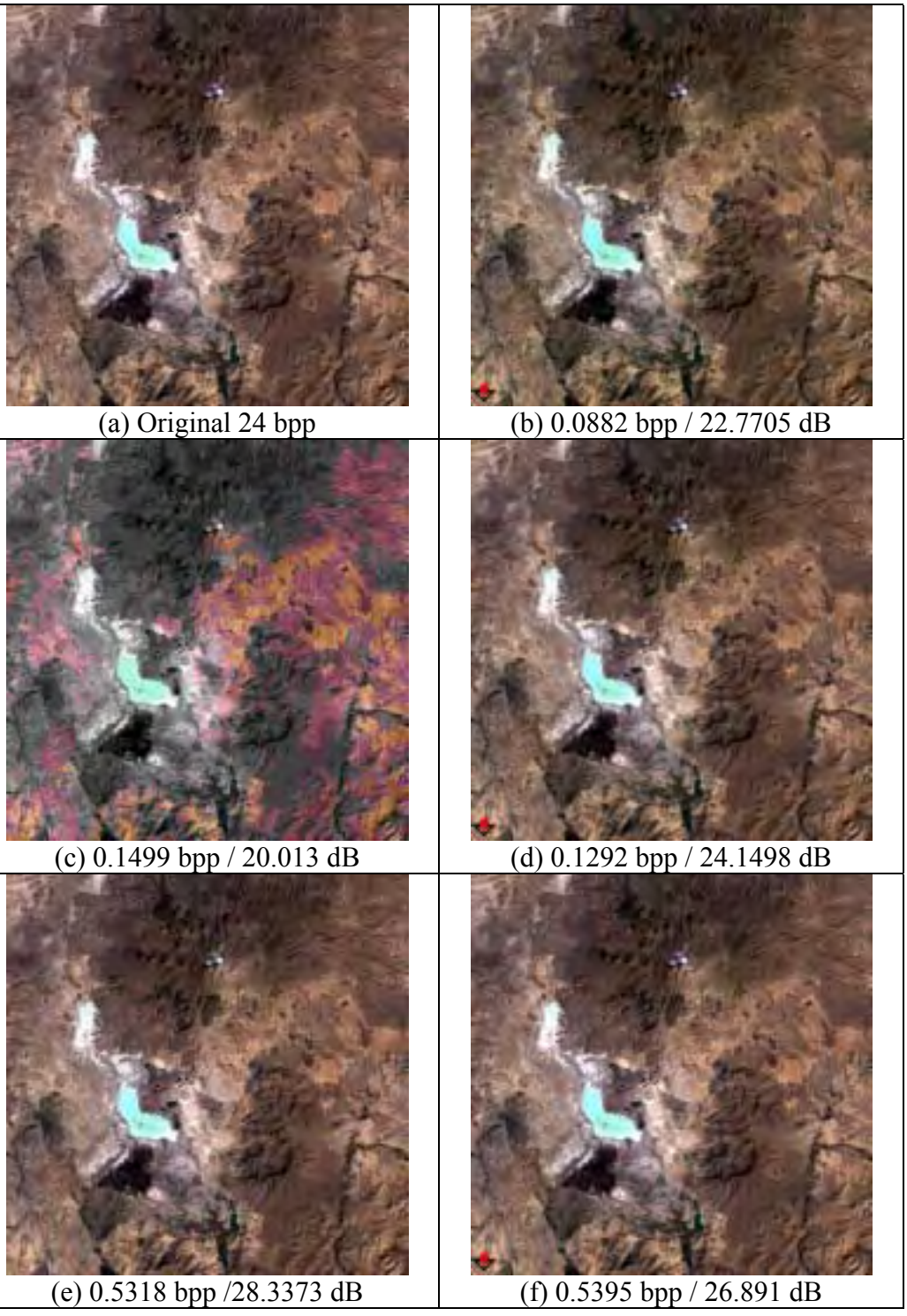
$$\mathrm{MSE} = \frac{1}{3 n_1 n_2} \sum_{m \in \{R,G,B\}} \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \left[x_m(i,j) - \hat{x}_m(i,j) \right]^2 \tag{4.3}$$

for RGB images.

where $x(i,j)$ and $\hat{x}(i,j)$ is the original and the recovered image respectively.

Figure 4.5-7 show the output obtained by conventional JPEG algorithm[60] and GEZGİN output. Images on the right hand side are provided from the GEZGİN engineering model[1] with the courtesy of TÜBİTAK-BİLTEN. For comparison purposes output file sizes are kept nearly equal. Tile sizes of 256 × 256 are used. It is apparent that for high compression rates JPEG 2000 gives superior results compared to conventional JPEG algorithm. The blocking effects of JPEG seen in high compression rates are not present in JPEG 2000. For high compression ratios conventional JPEG exhibits color degradation. (see Figure 4.5-7)

[1] The output of the engineering model is obtained from the testbench.



(a) Original 24 bpp

(b) 0.0882 bpp / 22.7705 dB

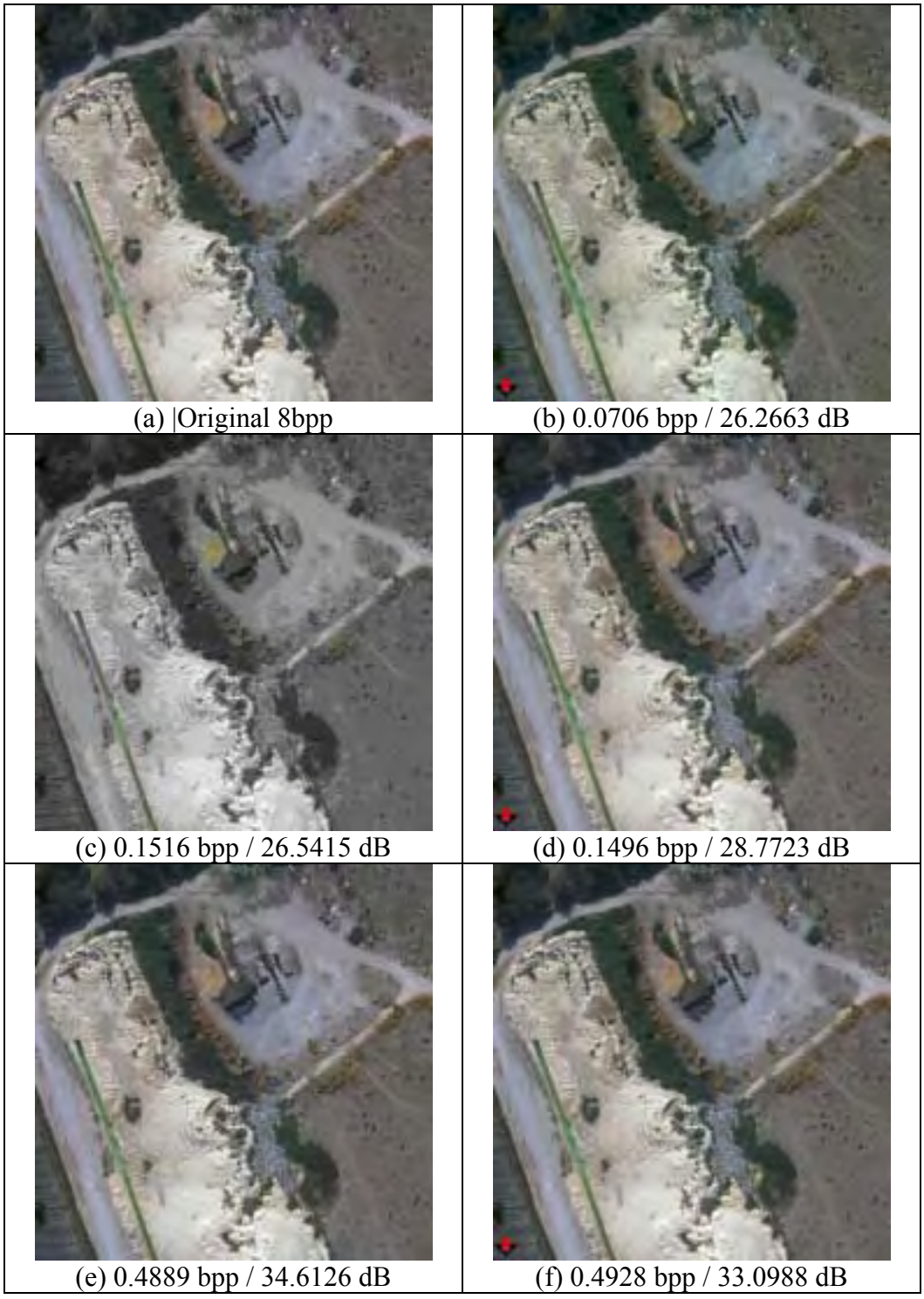
(c) 0.1499 bpp / 20.013 dB

(d) 0.1292 bpp / 24.1498 dB

(e) 0.5318 bpp /28.3373 dB

(f) 0.5395 bpp / 26.891 dB

Figure 4.5 The outputs of conventional JPEG and the outputs of GEZGİN for the original image of ERCIYES 2048×2048 24 bpp RGB. Images on the right hand side are provided from the GEZGİN test bench with the courtesy of TÜBİTAK-BİLTEN.

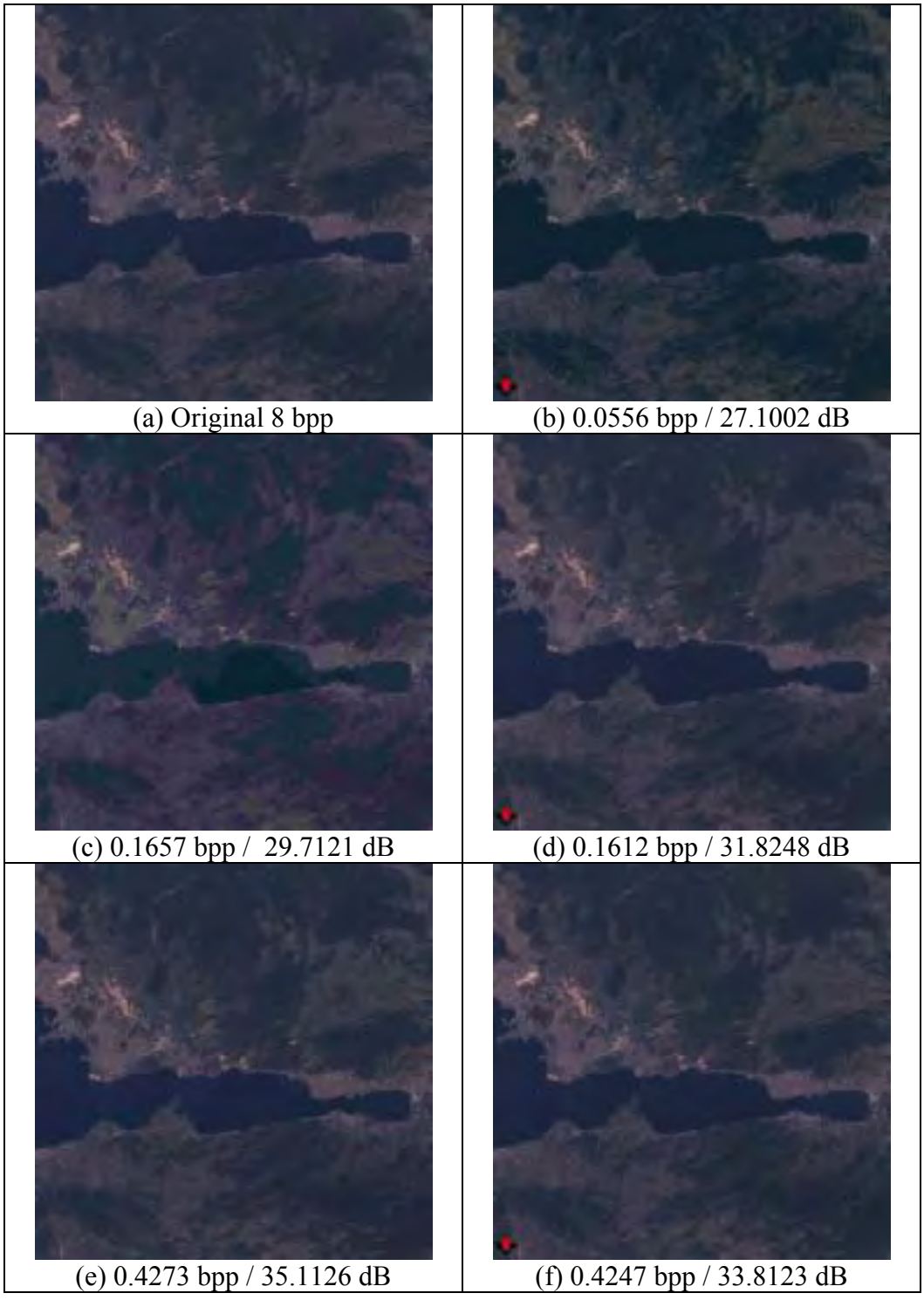


(a) |Original 8bpp | (b) 0.0706 bpp / 26.2663 dB

(c) 0.1516 bpp / 26.5415 dB | (d) 0.1496 bpp / 28.7723 dB

(e) 0.4889 bpp / 34.6126 dB | (f) 0.4928 bpp / 33.0988 dB

Figure 4.6 The outputs of conventional JPEG and the outputs of GEZGİN for the original image of MERSIN 2048×2048 24 bpp RGB. Images on the right hand side are provided from the GEZGİN test bench with the courtesy of TÜBİTAK-BİLTEN.



(a) Original 8 bpp | (b) 0.0556 bpp / 27.1002 dB

(c) 0.1657 bpp / 29.7121 dB | (d) 0.1612 bpp / 31.8248 dB

(e) 0.4273 bpp / 35.1126 dB | (f) 0.4247 bpp / 33.8123 dB

Figure 4.7 The outputs of conventional JPEG and the outputs of GEZGİN for the original image of GOLCUK 2048×2048 24 bpp RGB. Images on the right hand side are provided from the GEZGİN test bench with the courtesy of TÜBİTAK-BİLTEN.

Table 4.8 shows the lossless compression performance of JPEG 2000 and TIFF for various images. JPEG 2000 operation mode for the example is 3 levels of decomposition with color transform, tile-based processing with tile sizes of 256×256, and run-mode entropy coding with fixed code block sizes of 32×32, which is the same as that of GEZGİN.

Table 4.8 Lossless compression performance of PNG, LS and JPEG 2000. JPEG2000 results are obtained using GEZGİN Simulator.

Image	Compression (bpp)		
	PNG[61]	LS[62]	JPEG 2000[19]
Erciyes 2048,24b,RGB	*15.5725*	*15.5679*	*15.1292*
Mersin 2048,24b,RGB	*12.2316*	*11.3179*	*11.0489*
Gölcük 2048,24b,RGB	*11.3176*	*10.2187*	*9.8127*
BİLTEN Staff 2048,24b,RGB	*10.1375*	*8.6200*	*7.2175*
Denver 512,24b,RGB	*17.9659*	*18.1640*	*11.0174*
Lena 512,24b,RGB	*14.5320*	*13.6047*	*13.7630*

4.3 Results

4.3.1 Levels of Sub-band decomposition

Lossy compression is achieved by discarding specific sub-bands and applying quantization to the filter coefficients. For different levels of sub-band decomposition, bit discarding and sub-band omission is applied and PSNR values are reported. In all figures Si, $0 \leq i \leq J$ denotes that all HH, LH and LH outputs except the ones in the i greatest levels are omitted. For example S2 indicates that LL^3, LH^3, HL^3, HH^2, LH^2, HL^2, HH^2 are included but LH^1, HL^1, HH^1 are omitted for 3 levels of sub-band decomposition. In Figure 4.8, compression ratios achieved

for various number of decomposition levels and reconstruction resolutions are seen. Compression results are obtained from GEZGİN Test and Decoder Suite v1.0 [67].

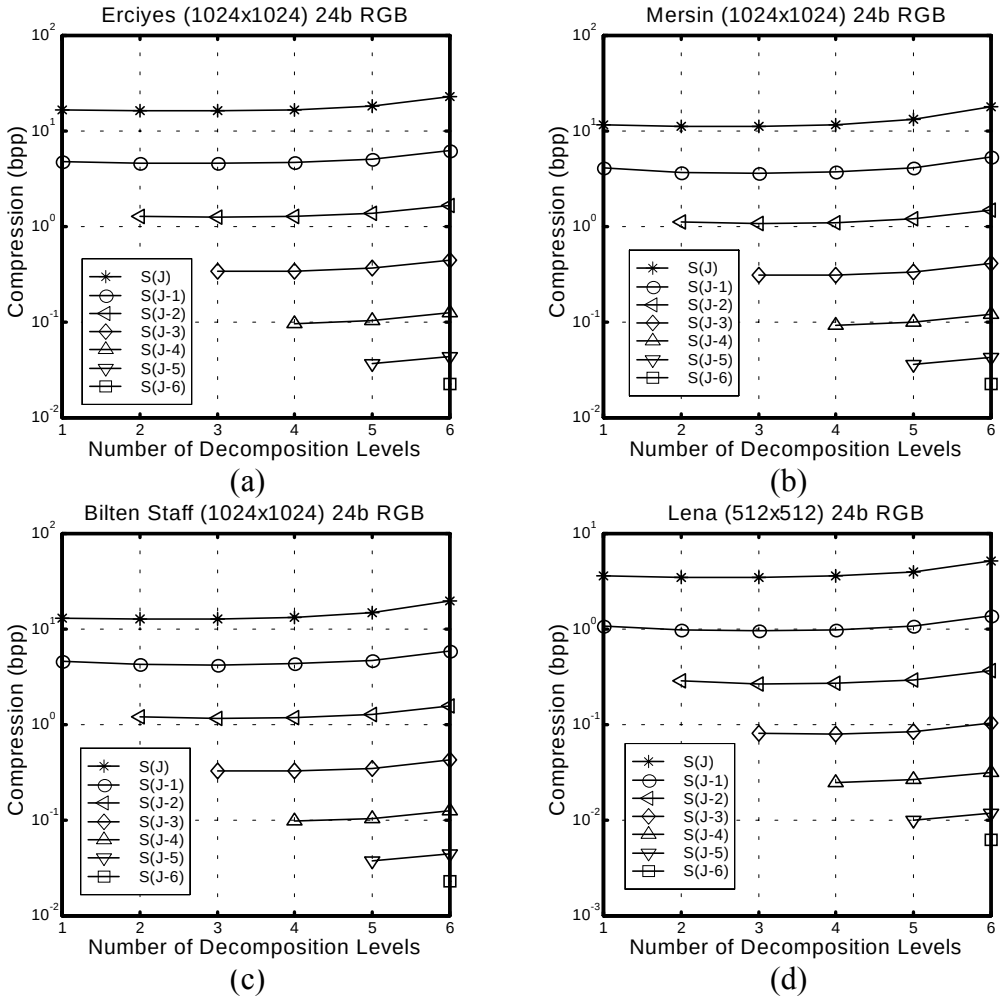


Figure 4.8 Compression ratios achieved for various number of decomposition levels and reconstruction resolutions.

4.3.2 Tile Size

For different tiles sizes 3 levels of sub-band decomposition is applied. Figure 4.9 shows the PSNR variation and compression achievements with tile size. Compression results are obtained from GEZGİN Test and Decoder Suite v1.0 [67]. It is apparent that smaller tile sizes lead to quality reduction, however this effect is

minor compared to sub-band exclusion. The quality gained by increasing the tile size from 256 to 512 is very small despite the huge memory requirement it brings, hence it is not preferred.

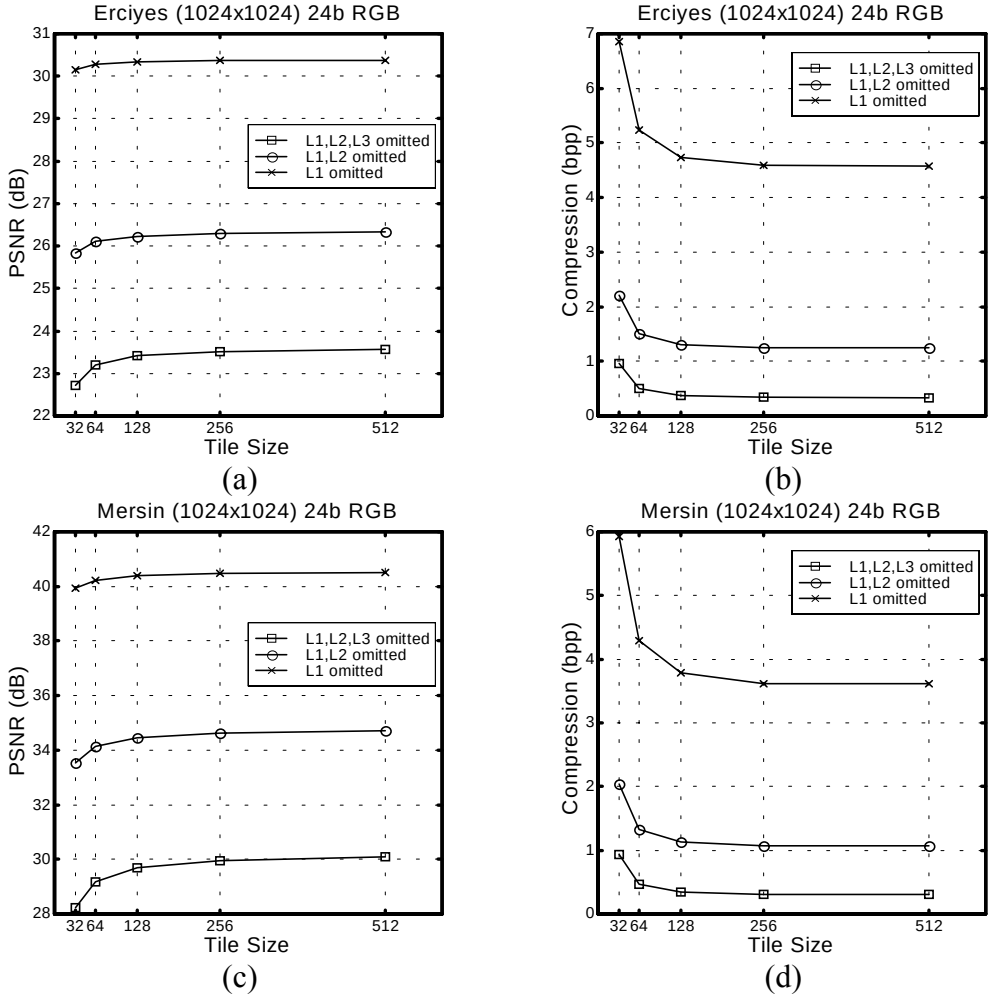


Figure 4.9 PSNR variation with tile size for 3 levels of sub-band decomposition and the corresponding compression achievements.

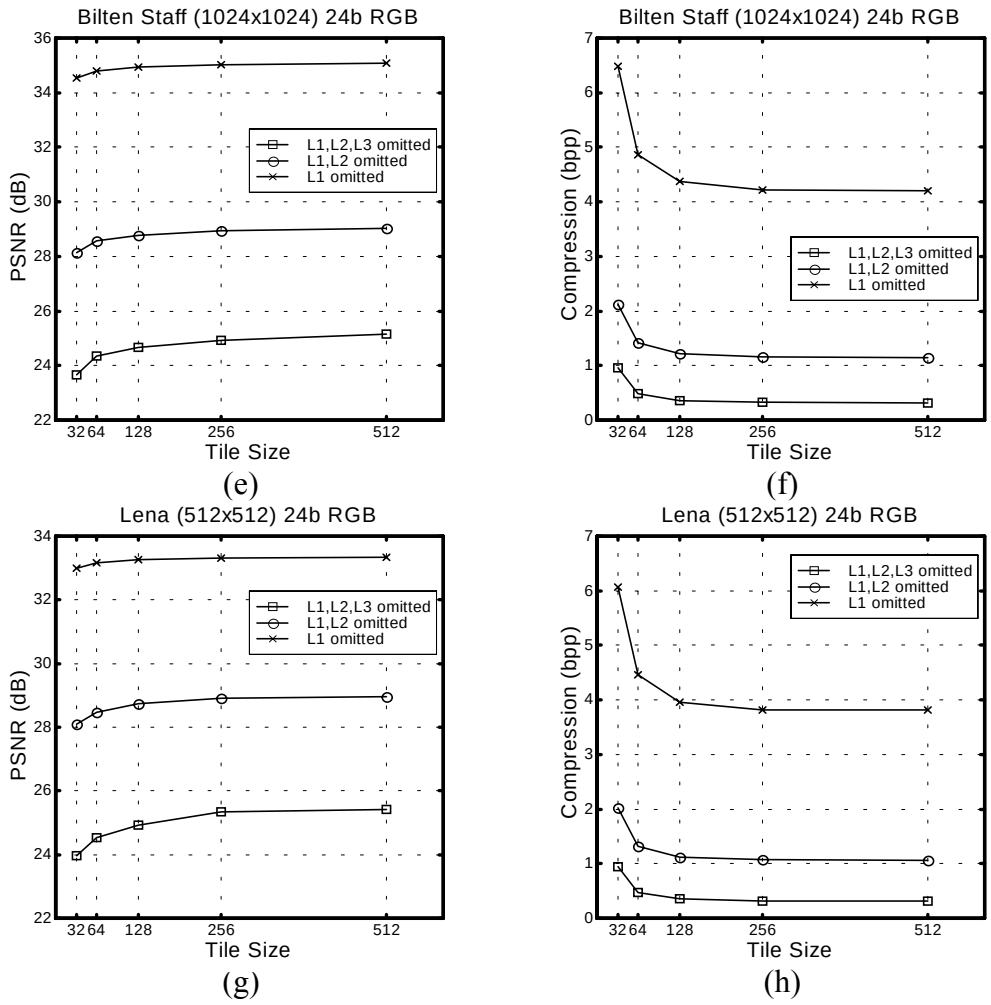


Figure 4.9 (Continued) PSNR variation with tile size for 3 levels of sub-band decomposition and the corresponding compression achievements.

4.3.3 Coefficient Truncation

Figure 4.10 shows the PSNR versus the number of bits discarded for various cases of sub-band decomposition. Up to a certain number of discarded bits, quality reduction of the recovered image is insignificant for S0, S1 and S2 cases, however the PSNR value rapidly decreases with beyond certain number of discarded bits. Note that further truncation of sub-band coefficients leads to vanishing benefit of sub-band inclusion. This is because the extra quantization of HL HH and LH sub-

bands introduces errors to these low energy sub-bands, and hence to the reconstructed image. As quantization increases, these extra noise becomes significant compared to -or may even become larger than- the noise eliminated by the inclusion of the sub-bands.

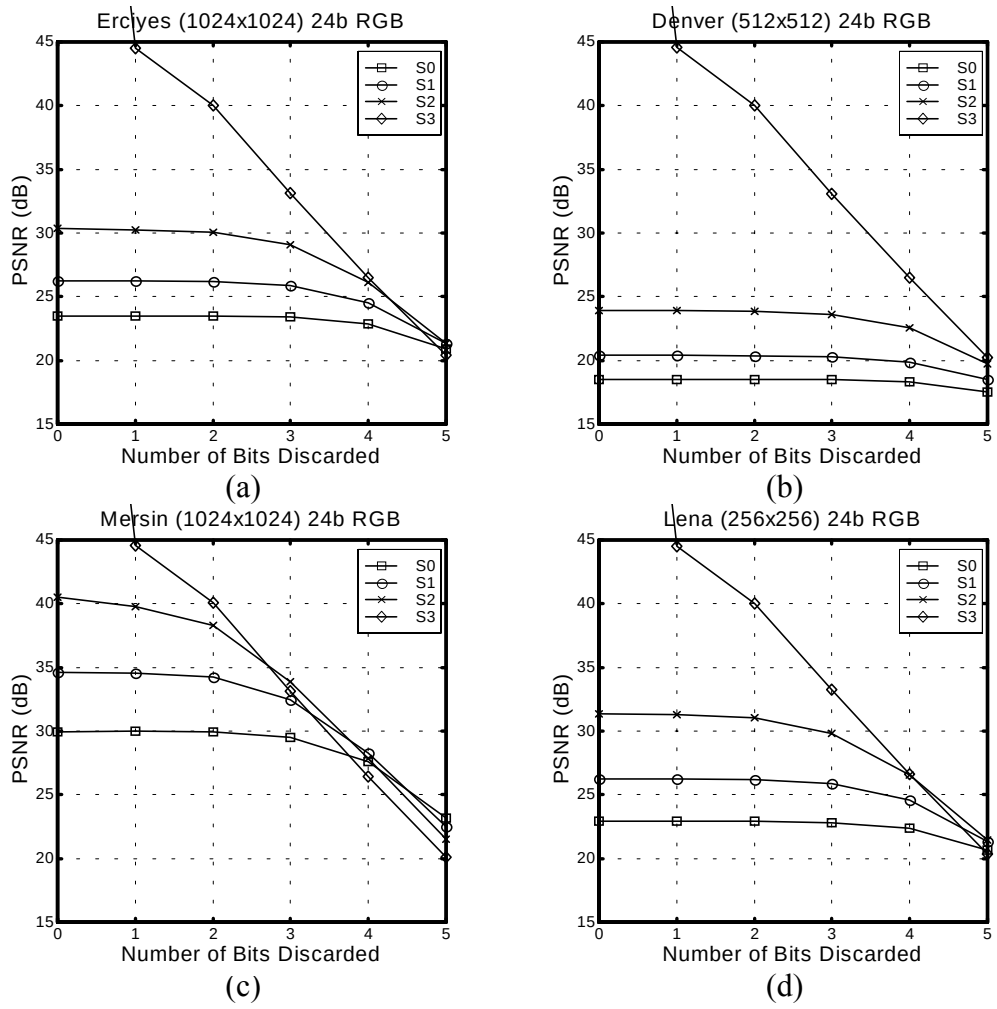


Figure 4.10 PSNR versus the number of bits discarded for various cases of sub-band decomposition.

In GEZGİN wavelet coefficients are computed with a precision of $B+\Gamma=10$ bits. Input bandwidth is 4×20 Mbps = 80 Mbps. From (3.32), taking into account the precision expansion, the required output bandwidth is calculated to be 150 Mbps.

Due to the limited output bandwidth available through the 16-bit HPI link, [63], the bit depth expansion constitutes a problem of transmission especially if these excess two bits are accommodated in 8-bits words. In this case, for each 16 bits a redundant transmission of 6 bits occurs resulting in a bandwidth expansion by a factor of two. Moreover it doubles the coefficient storage.

On the other hand rearrangement of coefficients in order to store $B+\Gamma$ bit data compactly in multiple-of-B bit storage unit may require excessive amount of logic or buffers in reconfigurable logic.

For these reasons it is preferred to apply bit discarding of $q=2$ to the wavelet coefficients prior to transmission. Figure 4.10 also shows the quality reduction in terms of PSNR due quantization of 2 bits. Note that high frequency sub-bands usually consist of very small coefficients centered around zero, i.e., these sub-bands contain less energy. Moreover, color transformed components which represent the difference of two input components, have smaller dynamic ranges. Therefore, specific sub-bands can be transmitted in 8 bits by clamping rather than quantizing, or both can be applied i.e. discarding only one bit and clamping coefficients in the range $[-128, 128)$.

4.3.4 Compression Time Experiments

Compression time is an important issue since each frame should be captured, compressed and transmitted before the next one arrives. Since the entropy coding and transmission time is proportional to the output size (capture time is constant and transform time depends only on the tile size) there is a trade-off between process time and image distortion. One of the missions of GEZGİN is to compress and transmit consecutive images which have overlapping areas. The time between two shots is a function of the satellite speed and the desired size of overlap area, and it is about 6.5 seconds for BILSAT-1. Therefore, it is important to identify modes of operation which fit the allotted time, while providing the highest quality.

Table 4. 9 Time required to process image and achieved distortion, bit-rate for various options available in GEZGİN

Tile Size	# Discarded Bits	Subbands Included	Fetch + Coding + Transmit Time (sec)	Capture + Transform Time (sec)	Overall Time (sec)	Quality (dB)	Compression (bpp)
With RCT							
256	*4*	*LL LH HL*	*1,78*	*3,76*	*5,54*	*23.87*	*0.229*
256	*3*	*LL LH HL*	*1,78*	*4,79*	*6,57*	*25.53*	*0.335*
256	*3*	*LL*	*1,78*	*2,10*	*3,88*	*23.78*	*0.120*
256	*3*	*LL LH HH*	*1,78*	*4,96*	*6,74*	*24.89*	*0.351*
256	*3*	*LL HH*	*1,78*	*3,59*	*5,37*	*24.01*	*0.244*
256	*2*	*LL*	*1,78*	*2,44*	*4,22*	*23.96*	*0.167*
256	*2*	*LL LH*	*1,78*	*4,16*	*5,94*	*24.82*	*0.317*
256	*2*	*LL HH*	*1,78*	*4,33*	*6,11*	*24.25*	*0.337*
256	*2*	*LL LH HL*	*1,78*	*5,87*	*7,65*	*25.93*	*0.468*
256	*4*	*LL LH HL HH*	*1,78*	*4,84*	*6,62*	*24.08*	*0.314*
256	*5*	*LL LH HL HH*	*1,78*	*3,65*	*5,43*	*19.91*	*0.230*
128	*4*	*LL LH HL*	*1,73*	*3,52*	*5,25*	*23.78*	*0.247*
128	*3*	*LL LH HL*	*1,73*	*4,55*	*6,28*	*25.44*	*0.354*
128	*3*	*LL*	*1,73*	*2,04*	*3,77*	*23.67*	*0.128*
128	*3*	*LL LH HH*	*1,73*	*4,72*	*6,45*	*24.79*	*0.242*
128	*3*	*LL HH*	*1,73*	*3,47*	*5,20*	*23.88*	*0.258*
128	*2*	*LL*	*1,73*	*2,38*	*4,11*	*23.84*	*0.175*
128	*2*	*LL LH*	*1,73*	*4,04*	*5,77*	*24.72*	*0.331*
128	*2*	*LL HH*	*1,73*	*4,21*	*5,94*	*24.13*	*0.351*
128	*2*	*LL LH HL*	*1,73*	*5,69*	*7,42*	*25.85*	*0.488*
128	*4*	*LL LH HL HH*	*1,73*	*4,61*	*6,34*	*24.00*	*0.339*
128	*5*	*LL LH HL HH*	*1,73*	*3,41*	*5,14*	*19.82*	*0.253*
Without RCT							
256	*4*	*LL LH HL*	*1,78*	*4,45*	*6,23*	*24.51*	*0.351*
256	*3*	*LL LH HL*	*1,78*	*5,59*	*7,37*	*25.70*	*0.351*

High compression ratios are obtained by bit discarding (quantizing) wavelet coefficients and sub-band exclusion. Applying a Reversible Color Transform (RCT) [19] prior to the DWT reduces the dynamic range of the wavelet coefficients, and hence higher compression rates are achieved [66]. RCT also speeds up the compression job, but added noise due to quantization is amplified (also due to the reduction in dynamic range). Table 4.9 shows the time required to capture, compress and transmit and the achieved distortion, bit-rate for various options available on GEZGİN. The test is performed on the image, Erciyes (2048×2048) 24b RGB. Figure 4.11 shows the PSNR reduction introduced due to RCT and the corresponding bit rate achievement for different cases. Compression results are

obtained from GEZGİN Test and Decoder Suite v1.0 [67]. Tile size is chosen as 256.

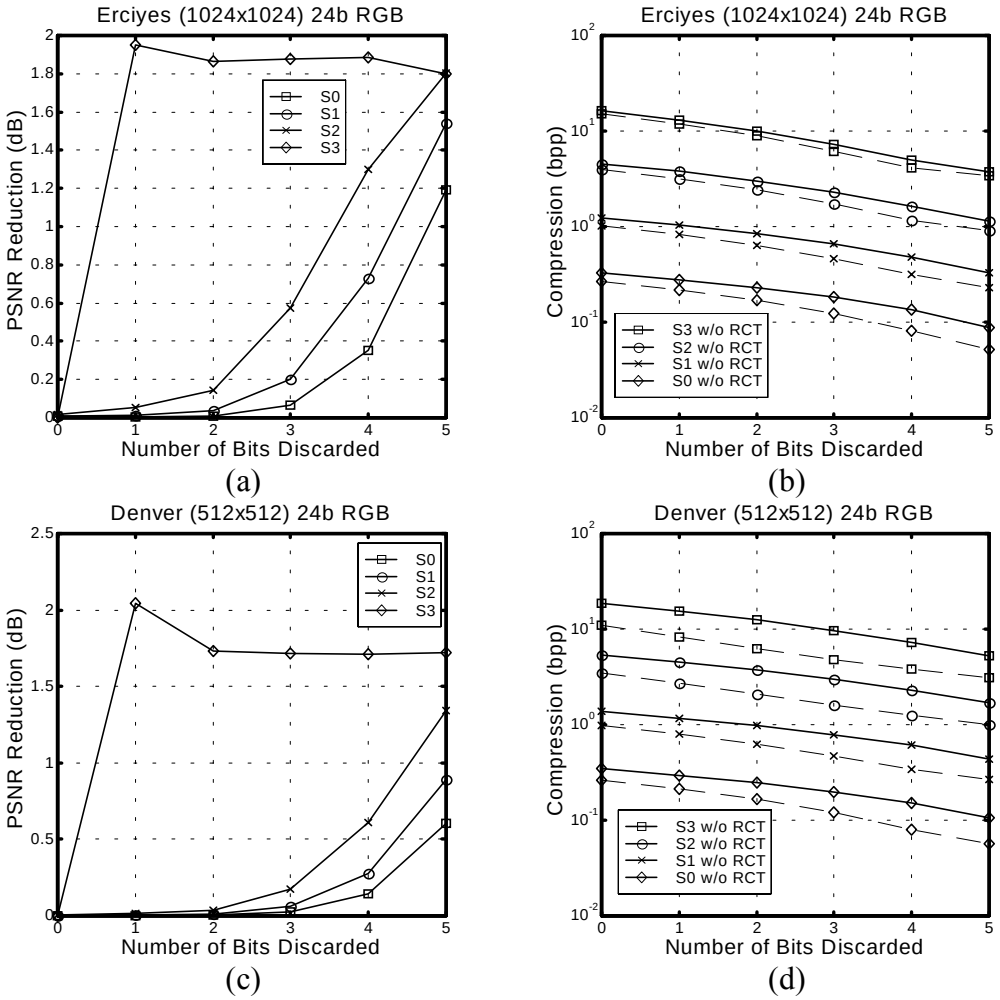


Figure 4.11 PSNR reduction introduced due to RCT and the corresponding bit rate achievement for different cases. Tile size is chosen as 256.

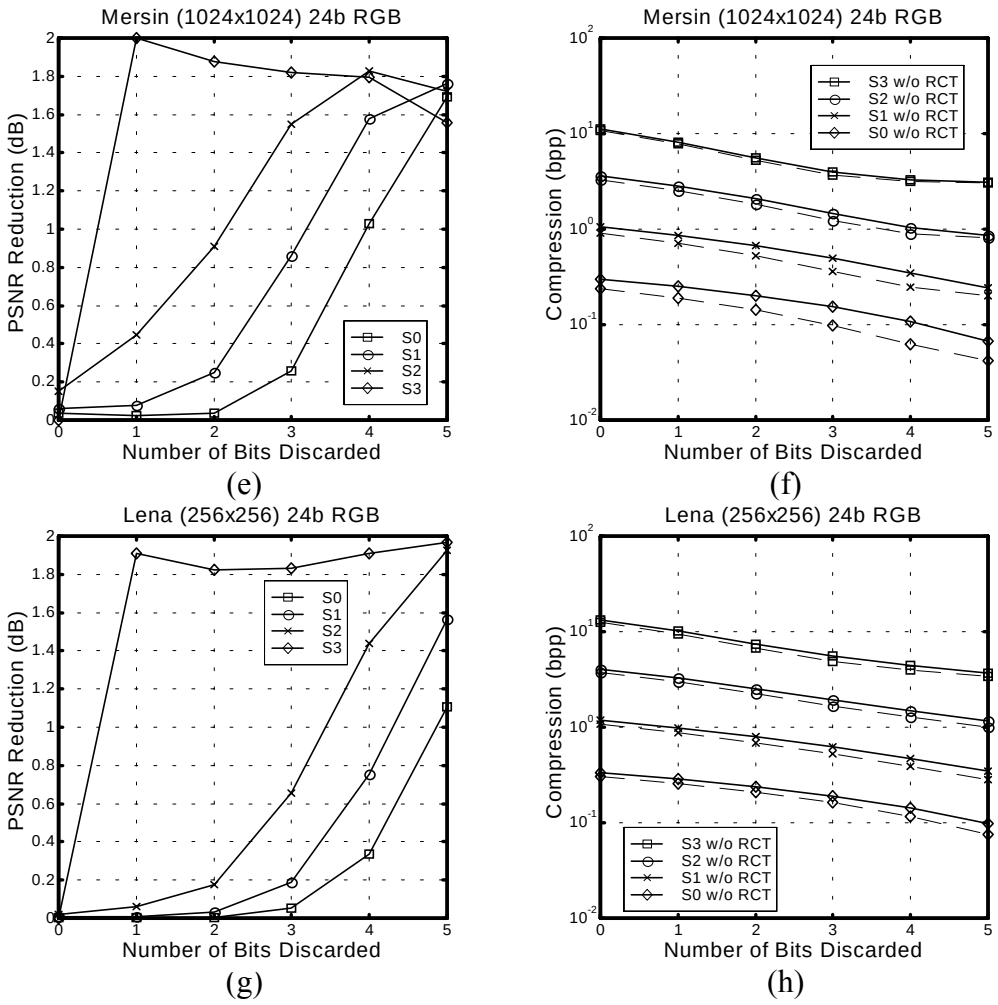


Figure 4.11 (Continued) PSNR reduction introduced due to RCT and the corresponding bit rate achievement for different cases. Tile size is chosen as 256.

4.3.5 Dynamic Range Expansion at the RCT output

In Section 3.3 the dynamic range expansion of color transformed pixels is discussed. Several experiments are done by clamping and/or quantizing color transformed pixels the images are supplied from [64]. Table 4.10 shows the results obtained. Images with asterisk are the aerial images from satellites. It is apparent that, for aerial images, applying clamping to the pixels give better results than bit discarding.

In GEZGİN applications it is preferred to apply clamping to these color components after RCT.

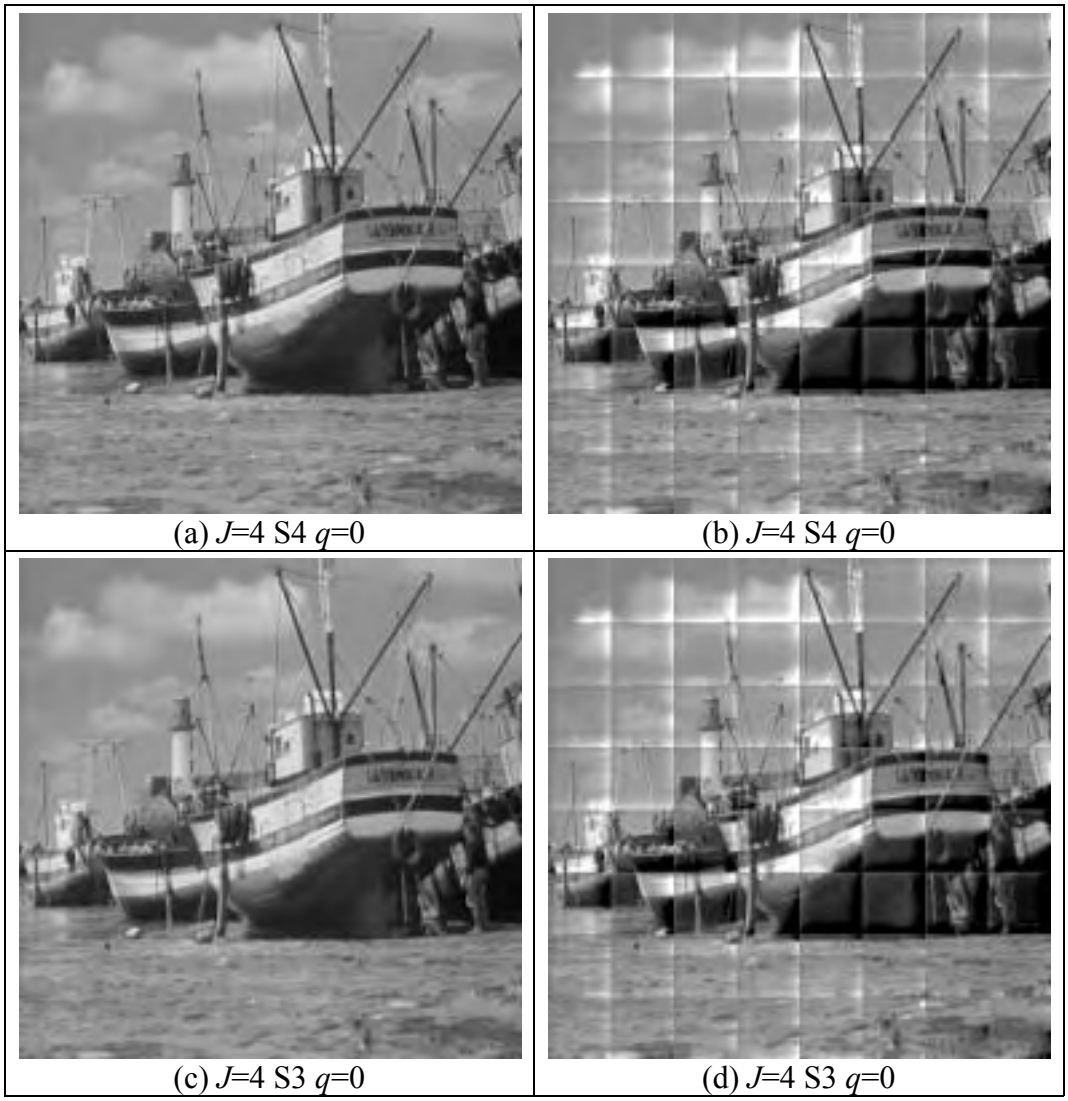
Table 4.10 Quality achievement of quantizing and clamping of color transformed samples

Image	S3		S2		S1		S0	
	Q (dB)	C (dB)	Q (dB)	C (dB)	Q (dB)	C (dB)	Q (dB)	C (dB)
1	*53.66*	*73.03*	*32.50*	*32.51*	*27.84*	*27.85*	*24.69*	*24.70*
2	*53.46*	*39.92*	*31.19*	*30.73*	*26.65*	*26.53*	*22.81*	*22.78*
3	*53.49*	*50.28*	*26.46*	*26.45*	*22.54*	*22.53*	*19.54*	*19.54*
4	*53.58*	*29.70*	*32.21*	*27.83*	*26.18*	*24.65*	*21.79*	*21.20*
5	*53.49*	*62.04*	*31.34*	*31.34*	*26.23*	*26.23*	*22.91*	*22.92*
6	*53.48*	*33.85*	*23.31*	*22.96*	*20.62*	*20.44*	*19.40*	*19.27*
7	*53.49*	*48.35*	*28.62*	*28.58*	*24.50*	*24.49*	*21.27*	*21.27*
8	*53.58*	*33.60*	*31.38*	*29.43*	*27.77*	*26.84*	*24.28*	*23.87*
*9**	*53.21*	*Inf*	*23.93*	*23.93*	*20.39*	*20.39*	*18.51*	*18.51*
10	*53.51*	*42.53*	*28.34*	*28.25*	*23.75*	*23.73*	*21.33*	*21.33*
11	*53.49*	*27.24*	*32.71*	*26.26*	*29.19*	*25.24*	*25.61*	*23.47*
*12**	*53.48*	*61.47*	*25.92*	*25.92*	*23.32*	*23.32*	*21.81*	*21.81*
13	*53.41*	*56.14*	*25.76*	*25.76*	*23.07*	*23.07*	*21.30*	*21.30*
14	*53.48*	*99.31*	*33.81*	*33.83*	*28.00*	*28.00*	*23.94*	*23.95*
*15**	*53.51*	*Inf*	*30.31*	*30.33*	*26.22*	*26.23*	*23.49*	*23.49*
*16**	*53.48*	*Inf*	*40.23*	*40.33*	*34.52*	*34.55*	*29.89*	*29.92*
*17**	*53.48*	*Inf*	*27.14*	*27.15*	*23.77*	*23.77*	*21.64*	*21.64*
*18**	*53.56*	*57.85*	*31.68*	*31.69*	*27.95*	*27.96*	*25.89*	*25.89*
*19**	*53.48*	*90.96*	*30.25*	*30.26*	*25.28*	*25.28*	*22.02*	*22.02*
*20**	*53.48*	*Inf*	*29.26*	*29.27*	*26.75*	*26.76*	*25.21*	*25.22*
*21**	*53.48*	*Inf*	*31.43*	*31.45*	*28.80*	*28.81*	*27.35*	*27.36*
*22**	*53.48*	*Inf*	*38.90*	*39.08*	*34.40*	*34.42*	*31.75*	*31.77*

4.3.6 Blocking Artifacts

Effect of tiling to quality degradation is examined for several cases. Figure 4.12 shows recovered images. For visual purposes tile size is kept small in order to recognize blocking effects and degradations. Tile sizes are chosen as 64×64. As the number of discarded HH, HL and LH sub-bands increases the blocking effect grows, however bit discarding does not have much impact on amount of blocking artifacts.

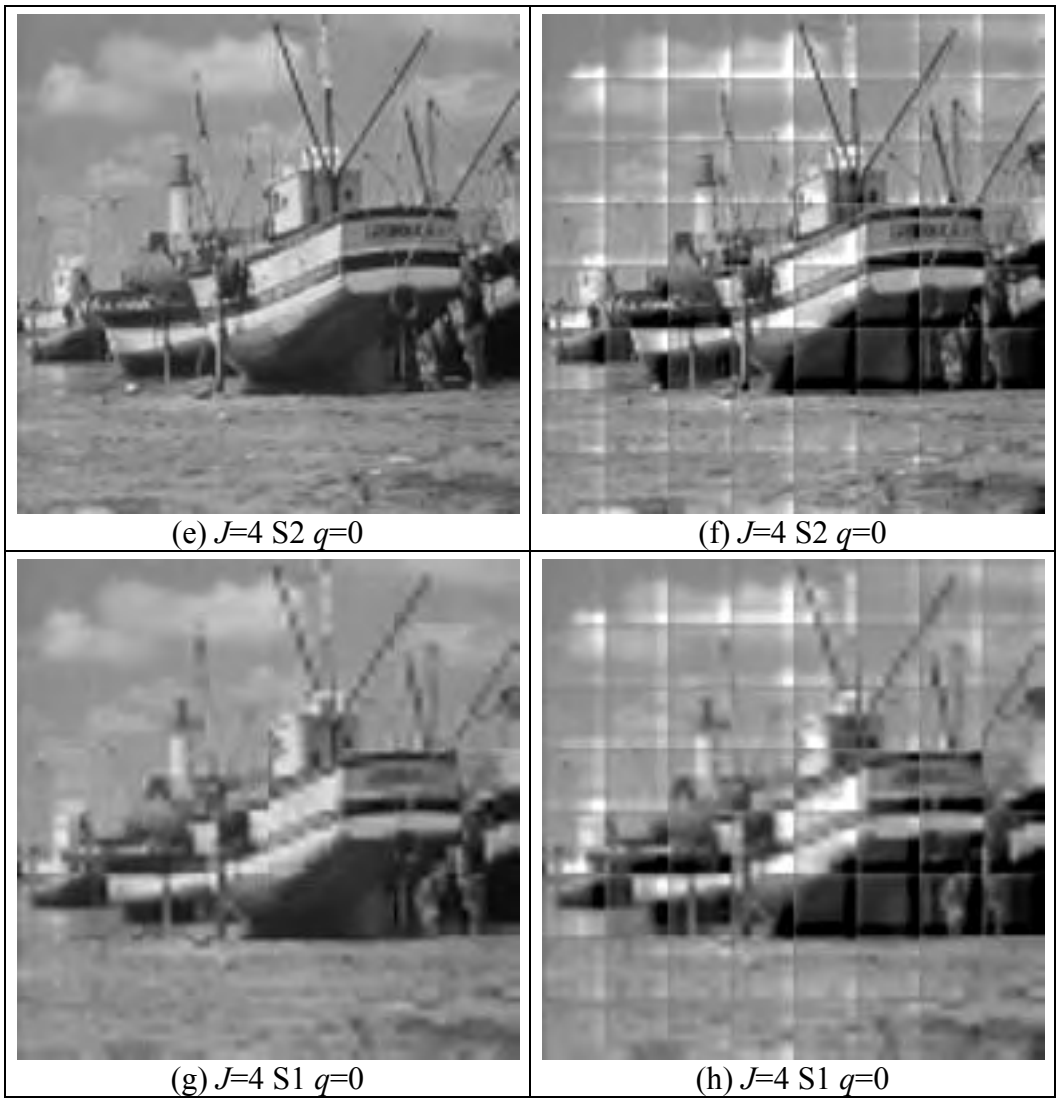
Figure 4.12 also shows the comparison of symmetric extension and zero padding. Images on the right side are subjected to zero padding, left side images are filtered using symmetric extension. It is apparent that zero padding does not allow for perfect reconstruction and that the symmetric extension is very beneficial since it results in considerably reduced blocking artifacts compared to zero padding. Although symmetric extension brings extra logic to the architecture, its application is vital.



(a) J=4 S4 q=0 (b) J=4 S4 q=0

(c) J=4 S3 q=0 (d) J=4 S3 q=0

Figure 4.12 Effect of tiling to quality degradation. Images on the right side are subjected to zero padding, left side images are filtered using symmetric extension.

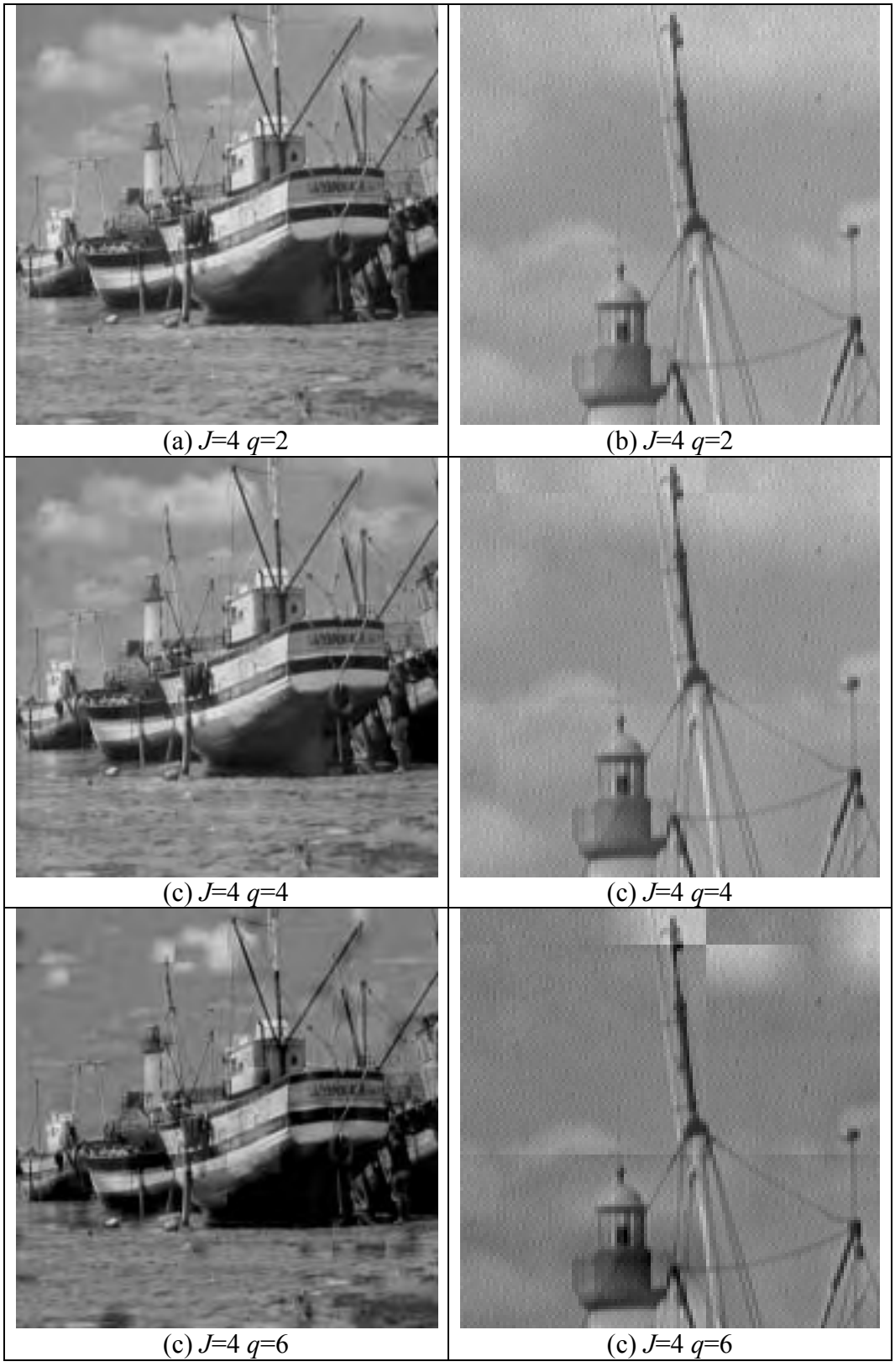


(e) J=4 S2 q=0 (f) J=4 S2 q=0

(g) J=4 S1 q=0 (h) J=4 S1 q=0

Figure 4.12 (Continued) Figure 4.12 Effect of tiling to quality degradation. Images on the right side are subjected to zero padding, left side images are filtered using symmetric extension.

Figure 4.13 shows the effect of quantization and Figure 4.12 shows the effect of sub-band exclusion. It is apparent that the effect of quantization is very minor and the tile information is contained in HL LH and HH sub-bands.



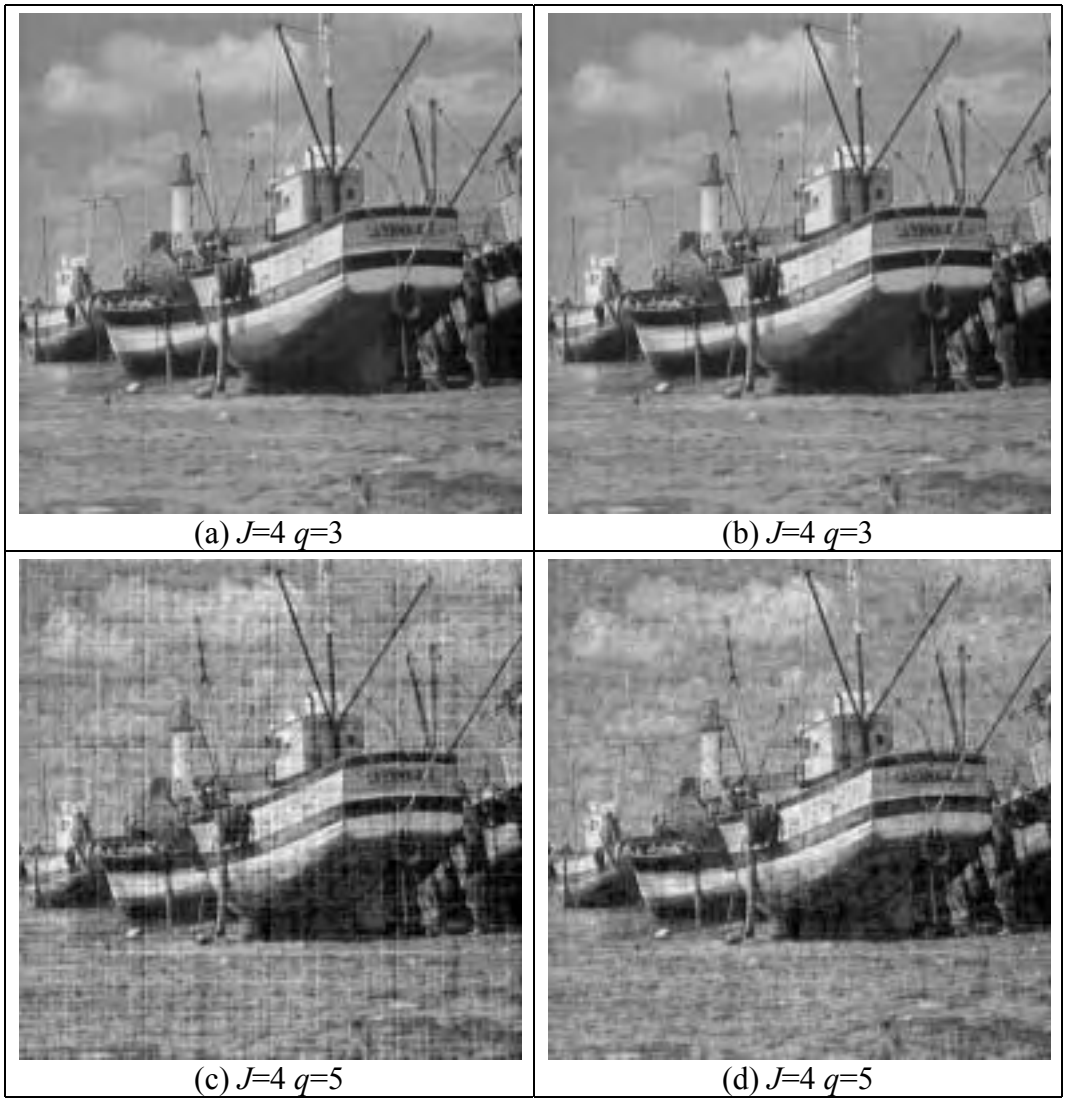
(a) J=4 q=2 (b) J=4 q=2

(c) J=4 q=4 (c) J=4 q=4

(c) J=4 q=6 (c) J=4 q=6

Figure 4.13 The effect of quantization of LL sub-band.

Although the 5/3 filter provides for reversibility, due its non-linear property [65][12] further quantizations result in poor results (see Figure 4.14). This can be reduced by applying dead-zone quantizing to the filter coefficients, rather than bit discarding. In GEZGİN only bit discarding is present, however it is observed that, in recovery, applying DC adjustment (i.e. setting a zero DC by subtraction) to truncated coefficients, provides improvements. In Figure 4.14, images (a), (c), and (e) are obtained by only truncation in coding and multiplication by 2^q in decoding, while images (b), (d) and (f) are obtained by applying DC adjustment to the truncated coefficients before multiplication by 2^q.



(a) J=4 q=3 (b) J=4 q=3

(c) J=4 q=5 (d) J=4 q=5

Figure 4.14 Quantization of the sub-bands. Images on the left are obtained by only truncation in coding and multiplication by 2^q in decoding, while images on the right obtained by applying DC adjustment to the truncated coefficients before multiplication by 2^q.

(e) J=4 q=7

(f) J=4 q=5

Figure 4.14 (Continued) Quantization of the sub-bands. Images on the left are obtained by only truncation in coding and multiplication by 2^q in decoding, while images on the right obtained by applying DC adjustment to the truncated coefficients before multiplication by 2^q.

CHAPTER 5

CONCLUSION

In this thesis, an architecture and an FPGA implementation of the two dimensional discrete wavelet transformation (DWT) is presented, for applications where row-based raw image data is streamed in at high bandwidths and local buffering of the entire image is not feasible. The architecture is especially suited for multi-spectral imager systems, such as on board an imaging satellite, however can be used in any application where time to next image constraints require real-time processing of multiple images.

The proposed hardware has been implemented on an FPGA and is part of a JPEG 2000 compression system designed as a payload for a low earth orbit (LEO) micro-satellite, which will be launched in September 2003. The fundamental mission of the system is to process (compress) the output of digital imaging sensors in real-time, as the image data is output from the sensors, while storing only a small portion of the incoming image stream at any given time. The task of the processor presented in this work is to accomplish required transforms (RCT and DWT) and transmit data efficiently with a latency as small as possible.

In applications requiring high bandwidth processing of images, internal storage utilization is inevitable. Large sized multi-spectral images are partitioned into tiles, which dramatically reduces the internal storage requirement at the expense of off-chip (local) storage, where a minimum required portion of the incoming image stream is stored. The latency that is introduced as the images stream through the

DWT processor and the amount of locally stored image data is a function of the image and tile size. For an $n_1 \times n_2$ size image processed using $(n_1/k_1) \times (n_2/k_2)$ sized tiles the latency is equal to the time elapsed to accumulate a $(1/k_1)$ portion of one image. In addition, a $(2/k_1)$ portion of each image is buffered locally.

The memory requirement depends on the tile size and the tile size in turn affects the quality of the compressed image and the processing delay of the wavelet transform. With the help of simulation data on several images including aerial and satellite pictures we arrived at a wavelet decomposition level of three, applied on tile sizes of 256×256, which, as reported, gives optimum results in terms of distortion and compression.

A literature survey on architectures implementing 2-D DWT and a comparison in from an FPGA implementation perspective are given. Although the proposed architecture is memory optimized, the internal memory required to transform tiles of such a large size imposes the utilization of SRAM blocks provided in FPGA devices. We propose an unfolded architecture in which each stage utilizes its own RAM block resulting in simpler logic and less routing. The stages handling each level use no scheduling logic, but simple counters which generate the address for RAM access. For three levels of wavelet decomposition with 5/3 filter, the proposed architecture compares favorably to existing architectures in terms of memory requirement, resources used, and computation time. To accommodate high bandwidth data, parallel processing of P DWT modules is proposed.

The symmetric extension at the tile boundaries is inevitable for perfect reconstruction. It is reported that symmetric extension is also vital for lossy compression. Therefore, although it brings extra logic and routing to the hardware, an architecture capable of handling boundaries is preferred. A method which employs time variant filtering is proposed in order to eliminate the need for time gap between consecutive tiles. This scheme provides for a regular gap-free tile acquisition.

In terms of compression efficiency, it is beneficial to apply, to the image components, a color transform which de-correlates the data. It is reported that the application of RCT results in a shorter encoding and transmission period, which provides for a better compression and quality performance within the time restrictions of overlap image capturing. The bandwidth expansion due to the color transformation of data is examined and several simulation results are presented. Since the bandwidth expansion results in an increase in local storage requirement, for some applications avoiding the extra storage may be desired. It is reported that, for aerial images, truncation of one bit from the resultant 9 bit representation via clamping rather than quantizing gives better results.

The proposed architecture inherently results in a bandwidth requirement at the output stage. As was the case in GEZGİN, the output link may not accommodate such high-bandwidths, and therefore, a simple modification to the architecture is proposed, by which the burst-full operation of the original structure is eliminated and a bandwidth of only 3/2 times that of the input is achieved. Experimental results for quantization and clamping of the coefficients and the restoration quality that is sacrificed due to truncation of bits are also given.

The proposed hardware has been implemented on a XILINX XCV300EPQ240-6 IC to cooperate with a 32-bit floating-point Digital Signal Processor (DSP) which implements the entropy coding. The application requires a throughput of 80Mbits/s. The implementation can achieve a throughput of up to 160Mbit/s when the DWT processors are operated at 40MHz. The latency introduced is 0.105 sec (6.25% of total transmission time) for tile sizes of 256×256. The local storage size required for the tiling operation is 2 MB. The internal storage requirement is 1536 pixels. Equivalent gate count for the design is 292,447. The hardware has two parallel DWT processors (P=2). While storing color transformed coefficients in local storage for tiling operation clamping is used. Due to the limited output bandwidth available through the 16-bit HPI link between FPGA and DSP, it is preferred to apply bit discarding of q=2 to the wavelet coefficients prior to transmission.

In our study, we have focused on the design and the implementation of a 2-D DWT processor for a multi-spectral imaging application. The work presents optimizations for a configurable logic implementation and a possible imaging application environment. As a future work, the implementation of memory efficient line-based entropy coder can be studied. Alternatively, integration of entire JPEG 2000 algorithm on a single IC and the design of system-on-a-chip (SOC) of the entire image compression system, GEZGİN, arise as possible future studies of the thesis.

REFERENCES

[1] G. M. Davis and A. Nosratinia, "Wavelet-based image coding: an overview," *Applied and Computational Control, Signals, and Circuits,* vol. 1, no. 1, Spring 1998.

[2] W. Woods and S. D. O'Neil, "Subband coding of Images," *IEEE Transactions on Acoustic, Speech and Signal Processing*, vol. ASSP-34, pp. 1278-1288, October 1986.

[3] O. Rioul and M. Vetterli, "Wavelets and signal processing," *IEEE Signal Processing Magazine*, vol. 8, pp. 14-38, October 1991.

[4] I. Daubechies, "Orthonormal bases of compactly supported wavelets," *Communications on Pure and Applied Mathematics*, vol. XLI, pp. 909-996, 1988.

[5] C. Chui, *Wavelets: A Tutorial in Theory and Applications*. Academic Press, New York, 1992.

[6] M. Vetterli and J. Kovačević, *Wavelets and Subband Coding.* Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[7] J. Shapiro, "Embedded image coding using zero-trees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3345-3462, December 1993.

[8] A. Said and W. A. Pearlman, "A new, fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 243-250, June 1996.

[9] M. W. Marchellin and T. Fischer, "Trellis coded quantization of memoryless and Gauss-Markov sources," *IEEE Transactions Communications*, vol. 38, no. 1, pp. 82-93, January 1990.

[10] D. S. Taubman and M. W. Marcellin, *JPEG 2000: Image Compression Fundamentals, Standards and Practice.* Kluwer Academic Publishers, Norwell, Massachusetts, 2002.

[11] M. L. Hilton, B. D. Jawerth, and A. Segupta, "Compressing still and moving images with wavelets," *Multimedia Systems*, vol. 2, no. 5, pp. 218-227, December 1994.

[12] S. G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674-693, July 1989.

[13] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *Journal of Fourier Analysis and Applications,* 4(3), pp.245-267, 1998.

[14] W. Sweldens, "The lifting scheme: A custom-design construction of biorthogonal wavelets," *Applied and Computational Harmonic Analysis*, 3(2): 186-200, 1996.

[15] A. R. Calderbank, I. Daubechies, W. Sweldens, and B. Yeo, "Wavelet transforms that map integers to integers," *Applied and Computational Harmonic Analysis,* 5(3):332-369, 1998.

[16] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek, "An overview of JPEG 2000," *Proceedings of IEEE Data Compression Conference,* pp. 523-541, 2000.

[17] D. Santa-Cruz, and T. Ebrahimi, "A study of JPEG 2000 still image coding versus other standards," *X European Signal Processing Conference*, vol. 2, pp. 673-676, September 2000.

[18] A. Kaarna and J. Parkkinen, “Comparison of compression methods for multispectral images,” *Proceedings of the Nordic Signal Processing Symposium*, *NORSIG 2000*, pp. 251-254, June 2000.

[19] *ISO/IEC JTC 1/SC 29/WG 1 N1646R*, “JPEG2000 Image Coding System”.

[20] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, “Image coding using wavelet transform,” *IEEE Transactions on Image Processing*., vol. 1, no. 2, pp. 205-220, April 1992.

[21] “Information technology – coded representation of picture and audio information – lossy/lossless coding of bi-level images”, 14492 Final Committee Draft, *ISO/IEC JTC1/SC 29/WG1 N1359*, July 1999.

[22] A. Bradford, L. Gomes, G. Yüksel, and C. Özkaptan, “BİLSAT-1: A Low-cost, Agile, Earth Observation Micro-satellite for Turkey,” IAF2002, October 2002.

[23] N. İsmailoğlu, O.Benderli, I. Korkmaz, M. Durna, T. Kolçak, and Y. Ç. Tekmen, “A Real Time Image Processing Subsystem: GEZGİN”, *Sixteenth Annual Conference on Small Satellites*, August 12-15, 2002, Utah, US.

[24] *Aware Wavelet Transform Processor (WTP) Preliminary. Aware Inc.,* Cambridge, MA, 1991

[25] F. Fridman and E. S. Manolakos, “Distributed memory and control VLSI architectures for the 1-D discrete wavelet transform,” *Proceedings*, *IEEE VLSI Signal Processing VII,* 1994.

[26] R. Lang, E. Plesner, H Schröder, and A. Spray, “An efficient systolic architecture for the one-dimensional wavelet transform,” *Proceedings SPIE Conference on Wavelet Applications*, pp. 925-935, April 1994.

[27] A. Grzeszczak, M. K. Mandal, S. Panchanathan, and T. Yeap, “VLSI implementations of discrete wavelet transform,” *IEEE Transactions on VLSI Systems*, vol. 4, pp. 421-433, December 1996.

[28] S.Masud and J. V. McCanny, “Rapid design of biorthogonal wavelet transforms,” *Proceedings*, *IEEE Circuits Devices and Systems,* vol. 147, pp. 293-296, October 2000.

[29] J. T. Kim, Y. H. Lee, T. Isshiki, and H. Kunieda, “Scalable VLSI architectures for lattice structure-based discrete wavelet transform,” *IEEE Transactions on Circuits and Systems - II,* vol. 45, no. 8, pp. 1031-1043, 1998.

[30] M. Vishwanath, “The Recursive Pyramid Algorithm for the discrete wavelet transform,” *IEEE Transactions on Signal Processing,* vol. 42, no. 3, pp. 673-676, March 1994.

[31] G. Knowles, “VLSI architecture for the discrete wavelet transform,” *Electronics Letters,* vol. 26, no. 15, pp. 1184-1185, July 1990.

[32] M. Vishwanath, R. M. Owens, and M. J. Irwin, “VLSI architectures for the discrete wavelet transform,” *IEEE Transactions on Circuits and Systems - II,* vol. 42, no. 3, pp. 305-316, May 1995.

[33] C. Chakrabarti and M. Vishwanath, “Efficient realization of the discrete and continuous wavelet transforms: From single chip implementations to mapping on SIMD array computers,” *IEEE Transactions on Signal Processing,* vol. 43, pp. 759-771, March 1995.

[34] I. Urriza, J. I. Artigas, J. I. Garcia, L. A. Barragan, and D. Navarro, “VLSI architecture for lossless compression medical images using the discrete wavelet transform,” *Proceedings, Design, Automation and Test in Europe,* pp. 196-201, February 1998.

[35] Y. Kim, K. Jun, and K. Rhee, “FPGA implementation of subband image encoder using discrete wavelet transform,” *Proceedings, IEEE Region 10 Conference TENCON 99,* vol. 2, pp. 1335-1338, September 1999.

[36] A.S. Lewis and G. Knowles, "VLSI architecture for 2-D Daubechies wavelet transform without multipliers," *Electronics Letters,* vol. 27, no. 2, pp. 171-173, January 1991.

[37] C. Chakrabarti and C. Mumford, "Efficient realizations of analysis and synthesis filters based on the 2-D discrete wavelet transform," *Proceedings, IEEE ICASSP,* pp. 3256-3259, May 1996.

[38] C. Chakrabarti and M. Vishwanath, "Architectures for wavelet transforms: A survey," *Journal of VLSI Signal Processing,* vol.14, pp. 171-192, 1996.

[39] P. McCanny, S. Masud, and J. McCanny, "An efficient architecture for the 2-D biorthogonal discrete wavelet transform," *Proceedings, 2001 International Conference on Image Processing,* vol. 3, pp. 314-317, October 2001.

[40] C. Chen, Z. Yang, T. Wang, and L. Chen, "A programmable VLSI architecture for 2-D discrete wavelet transform," *Proceedings, International Symposium on Circuits and Systems,* pp. 619-622, May 2000.

[41] T. Park and S. Jung, "A high performance lattice architecture of 2D discrete wavelet transform for hierarchical image compression," *International Conference on Consumer Electronics,* pp. 352-353, June 2002.

[42] P. Wu and L. Chen, "An efficient architecture for two-dimensional discrete wavelet transform," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 11, no. 4, pp. 536-545, April 2001.

[43] T. Denk and K. Pahri, "Calculation of minimum number of registers in 2-D discrete wavelet transforms using lapped block processing," *Proceedings, International Symposium on Circuits and Systems,* vol. 26, pp. 1184-1185, July 1995.

[44] S. Paek, H. Jeon, and L. Kim, "Semi-recursive VLSI architecture for two dimensional discrete wavelet transform," *Proceedings, International Symposium on Circuits and Systems,* vol. 5, pp. 469-472, May 1998.

[45] F. Marino, "Two fast architectures for the direct 2-D discrete wavelet transform," *IEEE Transactions on Signal Processing,* vol. 49, no. 6, pp. 1248-1259, June 2001.

[46] P. E. Danielsson, "Serial/parallel convolvers," *IEEE Transactions on Computers,* vol. 33, pp. 1079-1086, 1988.

[47] *The Programmable Logic Data Book 2000,* Xilinx Inc., 2000, online available at http://www.xilinx.com.

[48] *Data Book 1998,* Altera Co., 1998, online available at http://www.altera.com

[49] M. J. Gormish, E. L. Schwartz, A. Keith, M. Boliek, and A. Zandi, "Lossless and nearly lossless compression for high quality images," *Proceedings of the SPIE/IS&T Conference on Very High Resolution and Imaging II*, vol. 3025, pp. 62-70, February 1997.

[50] C. Chrysafis, *Wavelet Image Compression Rate Distortion Optimizations and Complexity Reductions,* PhD Thesis, Department of Electrical Engineering, University of Southern California, March 2000.

[51] J. Reichel, M. Nadenau, and M. Kunt, "Row-based wavelet decomposition using the lifting scheme," *Proceedings of the Workshop on Wavelet Transforms and Filter Banks (WTFB 99)*, March 1999.

[52] C. M. Brislawn, "Preservation of subband symmetry in multirate signal coding," *IEEE Transactions on Signal Processing,* 43(12), pp. 1248-1259, December 1995.

[53] C. Herley, "Boundary filters for finite length signals and time varying filter banks," *IEEE Transactions on Circuits and Systems II,* vol. 42, no. 2, pp. 102-114, February 1995.

[54] J. Ritter and P. Molitor, "A partitioned wavelet-based approach for image compression using FPGA's," *IEEE 2000, Custom Integrated Circuits Conference*, November 1999.

[55] G. K. Wallace, "The JPEG still picture compression standard," *Comm. ACM*, vol. 34, pp. 30-44, 1991.

[56] W. Jiang and A, Ortega, "Lifting factorization-based discrete wavelet transform architecture design," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 5, May 2001.

[57] M. Ravasi, L. Tenze, and M. Mattavelli, "A scalable and programmable architecture for 2-D DWT decoding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 8, August 2002.

[58] *RC1000 Hardware Reference Manual*, Celoxica Ltd, 2001. (http://www.celoxica.com)

[59] XAPP152 "Power Estimator Tools v2.0" (http://www.xilinx.com/xapp/xapp-152.pdf), (http://www.xilinx.com/ise/power_tools/virtex_power_estimator_v16.xls)

[60] ISO/IEC 10918-1 and ITU-T Recommendation T.81. Information technology – digital compression and coding of continuous-tone still images: Requirements and guidelines, 1994.

[61] W3C, PNG (Portable Network Graphics) Specification, 1996

[62] ISO/IEC IS 14495, "Lossless and near-lossless compression of continuous-tone still images."

[63] *TMS320C6000 Peripherals – Reference Guide,* Texas Instruments Inc., 2001.

[64] University of Southern California – SIPI Image Database [Online]. http://sipi.usc.edu/services/database/Database.html

[65] M. D. Adams, *Reversible Wavelet Transforms and Their Application to Embedded Image Compression,* MSc Thesis, Department of Electrical and Computer Engineering, University of Waterloo, 1993.

[66] M. Domański and K. Rakowski, "Lossless and near-lossless image compression with color transformations," *Proceedings, International Conference on Image Processing*, pp. 454-457, October 2001.

[67] Taner Kolçak, *GEZGİN Test and Decoder Suite v1.0 - Documentation and Manual*, Technical Report, TÜBİTAK-BİLTEN, 2002.

[68] M. Aşkar and O. Tekinalp, "Turkish small satellite program: goals and policies," *2nd International Symposium of the International Academy of Astronautics*, pp. 369-372, April 2001.

APPENDIX A

DESIGN HIEARCHY

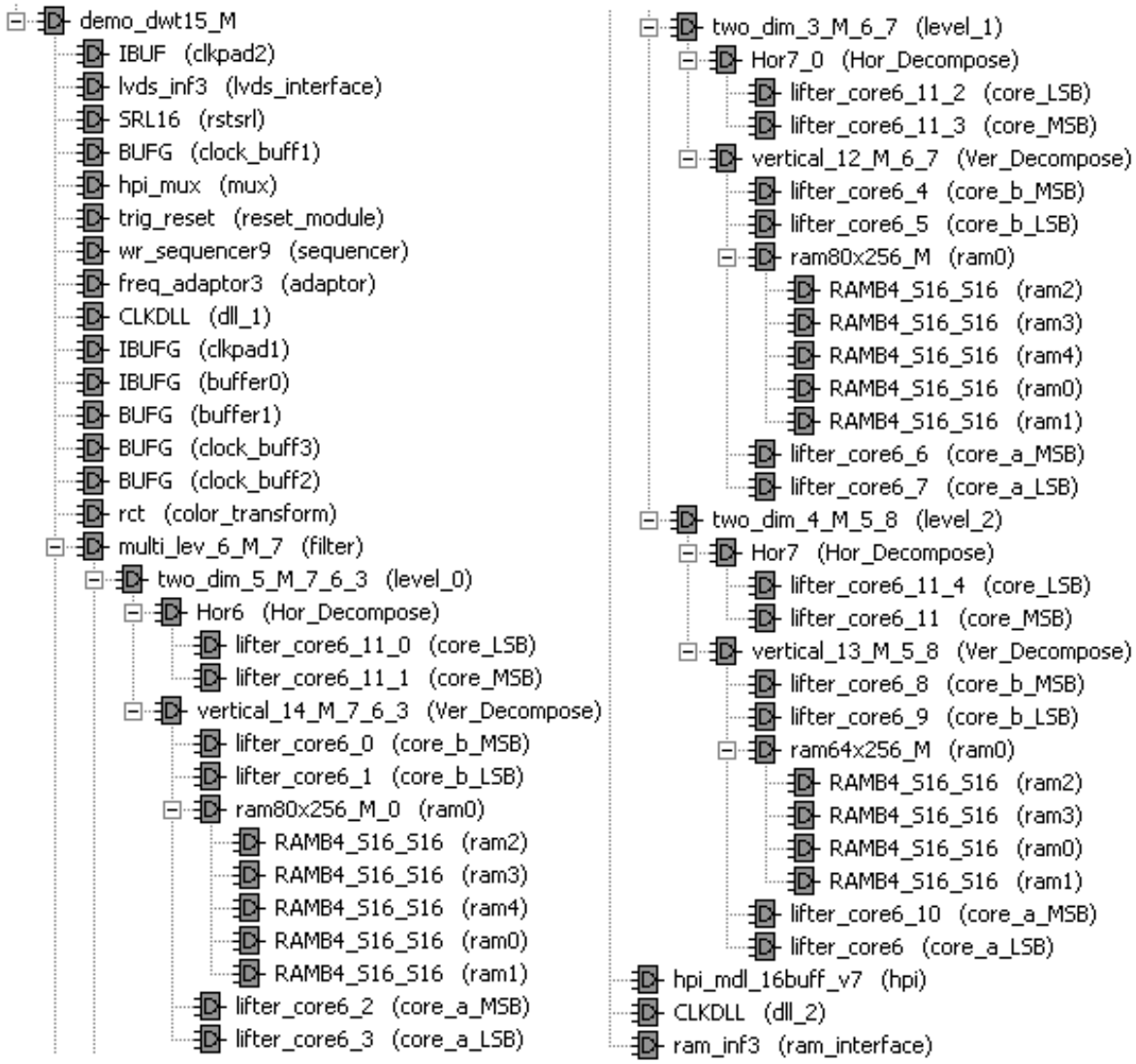


Figure A.1 The hierarchical structure of the design.

Figure A.1 shows the hierarchical structure of the design.

The description of the modules is as follows:

lvds_inf : This module receives the camera data form 4 serial links, discards the black stripes, translates it to 80MHz and sends to the next module (rct).

hpi_mux: This module multiplexes the level outputs of multi_lev module and translates the operation clock from 20MHz to 40MHz, and sends data to hpi_mdl_16_buff.

trig_reset: This module controls the reset signals distributed to the device and triggers the hardware when a TRIG signal from the cameras is received. TRIG signal indicates a new image has been shot and ready to be transmitted by the cameras.

wr_sequencer: This module handles the buffering (writing to local storage), tiling, reading back and sending data to the forthcoming modules. It receives data from rct and sends it to ram_inf for buffering, and sends the data that is read back from ram_inf to freq_adaptor .

freq_adaptor: this module receives data in 80MHz and sends it in 20MHz. It receives data from wr_sequencer and sends it to multi_lev.

rct: This module handles the DC-level shift operation and color transform. It receives data from lvds_inf and sends it to wr_sequencer.

multi_lev: This is the top module of the 2-D DWT, architecture. Data is received from frq_adaptor and send to hpi_mux lvds_inf.

two_dim: module handling 2-D DWT of 1 level. There are three instantiations of this module each belonging to a stage.

Hor: horizontal filtering and decomposition

Vertical: vertical filtering and decomposition

lifter_core: This module implements the arithmetic operations required for the lifting scheme of 5/3 filtering.

RAM80x256: This module contains five RAMB4_S16_16 ram block primitives, and used to store the samples required by stage 1 and 2.

RAM64x256: This module contains four RAMB4_S16_16 ram block primitives, and used to store the samples required by stage 3.

hpi_mdl_16buff[1]: this module is the HPI driver. It send the computed coefficients to the DSP through HPI link. It also contains buffer which compensate the temporary link stalls. It receives data from hpi_mux and outputs it from HPI pins.

ram_inf: this module is the SRAM driver for local storage. It receives data from wr_sequencer and outputs it from SRAM pins.

[1] This module is coded by Soner Yeşil.

APPENDIX B

VIRTEX-E RESOURCES

B.1 Architecture Overview

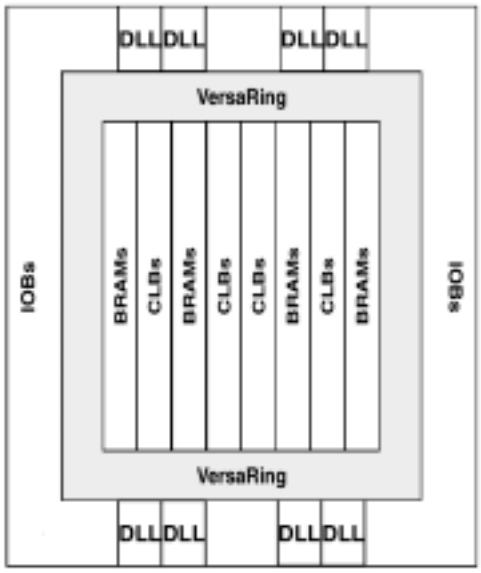


Figure B.1 Virtex-E Aarchitecture Overview.

Figure B.1 shows the Virtex-E Aarchitecture Overview. The FPGA comprises two major configurable elements: configurable logic blocks (CLBs) and I/O blocks (IOBs).

CLBs provide the functional elements for constructing logic, and IOBs provide the interface between the package pins and the CLBs. CLBs interconnect to a *general routing matrix*, which consists of an array of routing switches located at the intersections of horizontal and vertical routing channels.

The Virtex-E architecture also includes the following circuits that connect to the general routing matrix:

- Dedicated block memories of 4096 bits each.
- Clock Dlls for clock-distribution delay compensation and clock domain control
- Tri-state buffers associated with each CLB that drive dedicated segmentable horizontal routing resources.

Values stored in static memory cells control the configurable logic elements and the interconnect resources. These values load into the memory cells on power-up, and can reload if necessary to change the function of the device.

B.2 Configurable Logic Blocks (CLBs) and Slices

The basic building block of the Virtex-E CLB is the *logic cell*. a logic cell includes a 4-input function generator, carry logic, and a storage element. The output from the function generator in each logic cell drives both the CLB output and the D input of the flip-flop. Each Virtex-E CLB contains four logic cells as shown in Figure B.2. Each CLB is divided into two *slices*.

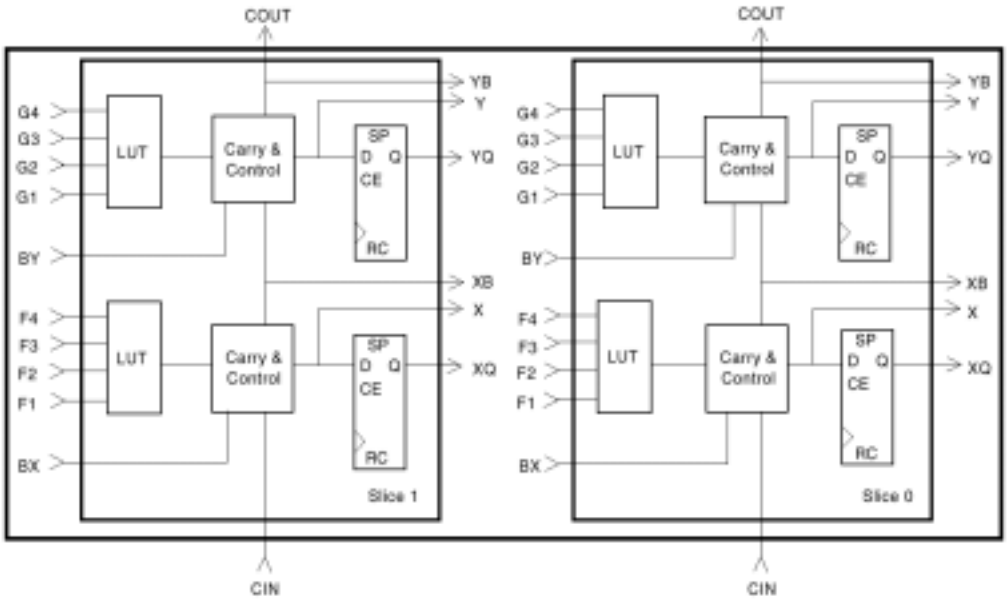


Figure B.2 Virtex-E CLB. Each Virtex-E CLB contains four logic cells and CLB is divided into two slices.

B.3 Look-up Tables (FGs)

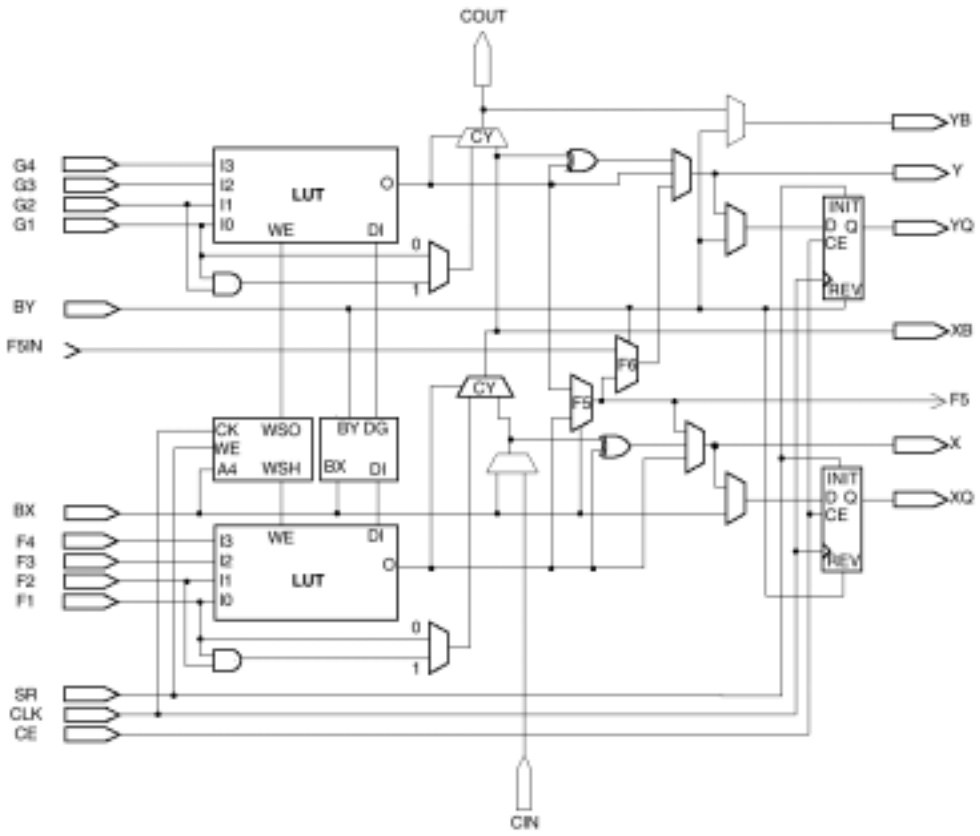


Figure B.3 The detailed schematic of a slice. A slice contains two LUTs, two DFFs, and one CY.

Virtex-E function generators are implemented as 4-input look-up tables (LUTs). In addition to operating as a function generator, each LUT can provide a 16 × 1-bit RAM, and a 16-bit shift register. Figure B.3 shows the detailed schematic of a slice having two LUTs.

B.4 Storage Elements (DFFs)

Figure B.3 shows the two storage elements provided in a slice. These elements can be either configured as edge-triggered D-type flip-flops or as level sensitive latches. The D inputs can be driven either by the function generators within the slice or directly from slice inputs, bypassing the function generators.

B.5 Arithmetic Logic (CYs)

Dedicated carry logic provides fast arithmetic carry capability for high-speed arithmetic functions. A CLB supports two separate *carry chains*, one per slice. They are indicated as “CY” in the Figure B.2.

The arithmetic logic also includes an XOR gate that allows a 2-bit full adder to be implemented within a slice.

B.6 Block SelectRAM (BRAM)

In Virtex-E FPGA provides with large Block SelectRAM memories. Block SelectRAM memory blocks are organized in columns (see Figure B.1) and inserted every 12 CLB columns. Each memory block is four CLBs high, and each memory column extends the full height of the chip. Each Block SelectRAM cell, as illustrated in Figure B.3 is a dual-ported 4096-bit RAM with independent control signals for each port. The data widths of the two ports can be configured independently.

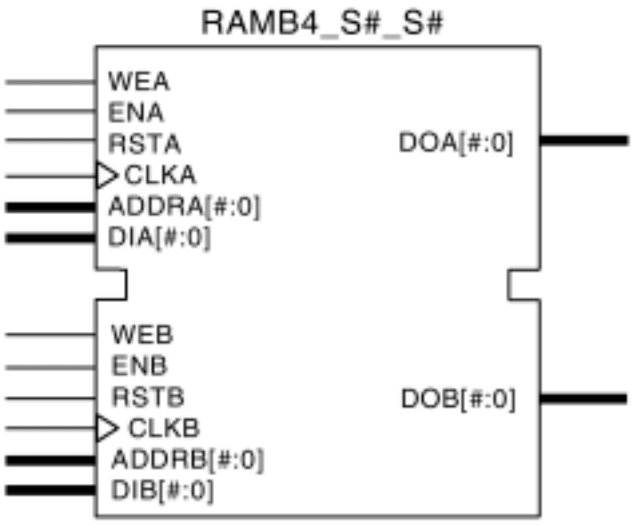


Figure B.4 Block SelectRAM cell.

Table B.1 shows the depth and width aspect ratios available for the RAM blocks.

Table B.1 Depth and width aspect ratios available for the RAM blocks.

Width	Depth	ADDR Bus	Data Bus
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

B.7 Digital Delay-Locked Loop (DLL)

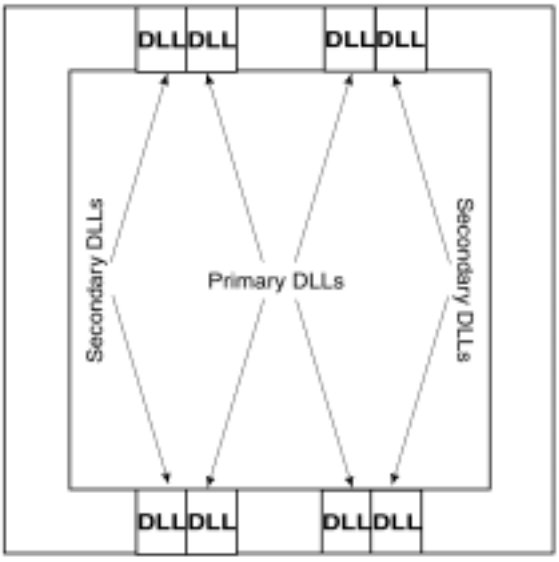


Figure B.5 Locations of the eight digital delay-locked loops (DLLs) in the device.

Virtex-E FPGA provides with eight delay-locked loops. four of them is located at the top and the other four is located at the bottom as shown in Figure B.4. The DLLs can be used to eliminate skew between the clock input pad and the internal clock input pins throughout the device. Each DLL can drive two global networks. The

DLL monitors the input clock and the distributed clock, and automatically adjusts a clock delay element.

In addition to eliminating clock-distribution delay, the DLL provides control of multiple clock domains. The DLL provides four quadrature phases of the source clock., and can double the clock or divide the clock by 2, 4, 8 or 16.

B.8 Global Clock Routing (GCLKs and GCLKIOBs)

In order to allow for high-speed, low-skew clock distribution, global routing resources are used. These are four dedicated clock nets (GCLK) and dedicated input pins (GCLKIOB). Each global clock net can drive all CLB, IOB, and block RAM clock pins. The global nets may only be driven by glocal buffers. There are four global buffers, one for each global net. Two global buffers are placed at the top center of the device and the remaining two at the bottom center as shown in Figure B.5. The input of the global buffer is selected either from global input pins or from signals in the general purpose routing.

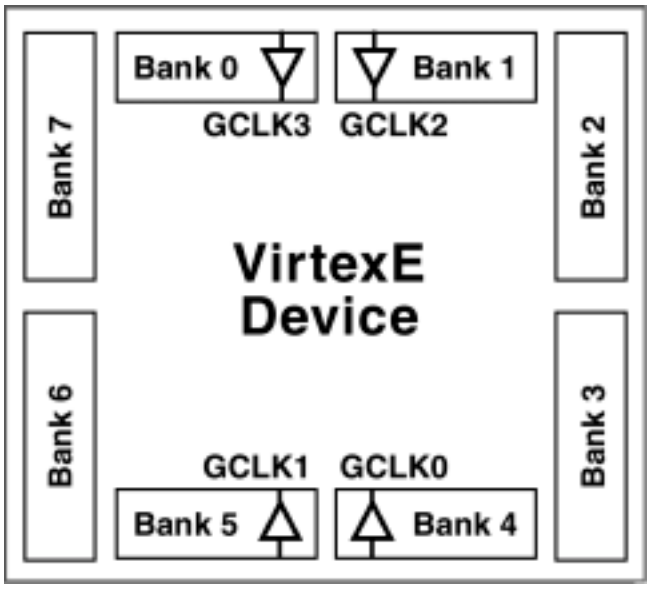


Figure B.6 Locations of the four global clock buffers (GCLKs) in the device.