HIGH-SPEED PIPELINE VLSI ARCHITECTURES FOR DISCRETE WAVELET TRANSFORMS

CHENG JUN ZHANG

A THESIS
IN
THE DEPARTMENT
OF
ELECTRICAL AND COMPUTER ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

MARCH 2012

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Cheng Jun Zhang

Entitled: High-Speed Pipeline VLSI Architectures for Discrete Wavelet Transforms

and submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY (Electrical & Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. N. Bhuiyan

_____ External Examiner
Dr. P.K. Meher

_____ External to Program
Dr. C.Y. Su

_____ Examiner
Dr. M.N.S. Swamy

_____ Examiner
Dr. W-P. Zhu

_____ Thesis Co-Supervisor
Dr. M.O. Ahmad

_____ Thesis Co-Supervisor
Dr. C. Wang

Approved by _____
Dr. J.X. Zhang, Graduate Program Director

April 5, 2012 _____
Dr. Robin A.L. Drew, Dean
Faculty of Engineering & Computer Science

ABSTRACT

High-Speed Pipeline VLSI Architectures for Discrete Wavelet Transforms

Cheng Jun Zhang, Ph.D.
Concordia University, 2012

The discrete wavelet transform (DWT) has been widely used in many fields, such as image compression, speech analysis and pattern recognition, because of its capability of decomposing a signal at multiple resolution levels. Due to the intensive computations involved with this transform, the design of efficient VLSI architectures for a fast computation of the transforms have become essential, especially for real-time applications and those requiring processing of high-speed data. The objective of this thesis is to develop a scheme for the design of hardware resource-efficient high-speed pipeline architectures for the computation of the DWT. The goal of high speed is achieved by maximizing the operating frequency and minimizing the number of clock cycles required for the DWT computation with little or no overhead on the hardware resources. In this thesis, an attempt is made to reach this goal by enhancing the inter-stage and intra-stage parallelisms through a systematic exploitation of the characteristics inherent in discrete wavelet transforms.

In order to enhance the inter-stage parallelism, a study is undertaken for determining the number of pipeline stages required for the DWT computation so as to synchronize their operations and utilize their hardware resources efficiently. This is achieved by optimally distributing the computational load associated with the various resolution levels to an optimum number of stages of the pipeline. This study has determined that employment of two pipeline stages with the first one performing the task of the first

resolution level and the second one that of all the other resolution levels of the 1-D DWT computation, and employment of three pipeline stages with the first and second ones performing the tasks of the first and second resolution levels and the third one performing that of the remaining resolution levels of the 2-D DWT computation, are the optimum choices for the development of 1-D and 2-D pipeline architectures, respectively. The enhancement of the intra-stage parallelism is based on two main ideas. The first idea, which stems from the fact that in each consecutive resolution level the input data are decimated by a factor of two along each dimension, is to decompose the filtering operation into subtasks that can be performed in parallel by operating on even- and odd-numbered samples along each dimension of the data. It is shown that each subtask, which is essentially a set of multiply-accumulate operations, can be performed by employing a MAC-cell network consisting of a two-dimensional array of bit-wise adders. The second idea in enhancing the intra-stage parallelism is to maximally extend the bit-wise addition operations of this network horizontally through a suitable arrangement of bit-wise adders so as to minimize the delay of its critical path.

In order to validate the proposed scheme, design and implementation of two specific examples of pipeline architectures for the 1-D and 2-D DWT computations are considered. The simulation results show that the pipeline architectures designed using the proposed scheme are able to operate at high clock frequencies, and their performances, in terms of the processing speed and area-time product, are superior to those of the architectures designed based on other schemes and utilizing similar or higher amount of hardware resources. Finally, the two pipeline architectures designed using the proposed scheme are implemented in FPGA. The test results of the FPGA implementations validate the feasibility and effectiveness of the proposed scheme for designing DWT pipeline architectures.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my deep gratitude to my supervisors, Dr. M. Omair Ahmad, and Dr. Chunyan Wang, for their support, encouragement, and invaluable guidance during this research. I am grateful to them for providing me freedom and motivation to explore new ideas in this research. I also thank them for spending countless long hours discussing the research in this thesis, and correcting and improving the writing of this thesis. The useful suggestions and comments provided by the members of the supervisory committee, Dr. M.N.S. Swamy, Dr. Weiping Zhu, and Dr. Chunyi Su, and by the External Examiner, Dr. Pramod K. Meher, as well as those of the anonymous reviewers of my journal papers, are deeply appreciated.

I would like to acknowledge the financial support provided by Concordia University and the Natural Sciences and Engineering Research Council (NSERC) of Canada, which were crucial to completing this research.

Table of Contents

List of Figures

List of Tables

List of Acronyms

1-D:	One-dimensional
2-D:	Two-dimensional
BRAM:	Block random access memory
CLB:	Configuration logic block
CPA:	Carry propagation adder
CWT:	Continuous wavelet transform
DFF:	D-type flip-flop
DRU:	Data recorder unit
DSP:	Digital signal processing
DSU:	Data scanning unit
DWT:	Discrete wavelet transform
FIR:	Finite impulse response
FPGA:	Field programmable gate array
FPS:	Frames per second
HH:	Highpass-highpass
HL:	Highpass-lowpass
IOB:	Input/output block

LH:	Lowpass-highpass
LL:	Lowpass-lowpass
LUT:	Look-up table
MAC:	Multiply-accumulate
MBPS:	Megabytes per second
N/A:	Not available
PU:	Processing unit
RPA:	Recursive pyramid algorithm
SIMD:	Single instruction multiple data
VLSI :	Very large scale integration

List of Symbols

a :	Dilation parameter
A :	Area
$\mathbf{A_I}$:	An array of input bits to a layer of MAC-cell network
$\mathbf{A_O}$:	An array of output bits from a layer of MAC-cell network
$A(m,n)$:	Highpass-highpass subband sample with index (m, n)
b :	Translation parameter
$B(m,n)$:	Highpass-lowpass subband sample with index (m, n)
$c(i)$:	Transformed scaling coefficients
$D(m,n)$:	Lowpass-highpass subband sample with index (m, n)
$f_{\max}$:	Maximum operational clock frequency
$f(.)$:	Function representing an input signal
$G(z)$:	Impulse response of a filter
$\mathbf{G}$:	Transform matrix representing a highpass filtering operation
g_i :	The ith coefficient of a highpass filter
$\mathbf{H}$:	Transform matrix representing of a lowpass filtering operation
h_i :	The ith coefficient of a lowpass filter
$H^{(P)}(k,i)$:	The (k,i)th coefficient of the 2-D filter P
I:	Number of pipeline stages
J :	Largest resolution level

j :	Index of resolution levels
L :	Length of filter
$\mathbf{L}^2(R)$:	Hilbert space
M :	Length of filter
n_x :	Number of samples computed in time span t_x (x=b or c)
N :	Number of input samples along one dimension
N_{ADD} :	Number of adders
N_{CLK} :	Number of clock cycles
N_j :	Number of samples at the jth resolution level
N_{MUL} :	Number of multipliers
N_{REG} :	Number of registers
p :	Counter for samples computed at resolution level 2
q :	Counter for samples computed at resolution levels higher than 2
Q(z):	Transform matrix for two-channel wavelet filters
S :	Matrix representation of an input signal
$S(m,n)$:	Lowpass-lowpass subband sample with index (m, n)
s_i :	The ith input sample
t_a :	Time span during which the first stage operates alone
t_b :	Time span during which the stages of a pipeline operate in parallel
t_c :	Time span during which the last stage operates alone
t_i :	Operating time period of the ith pipeline stage (i=1, 2, 3)
T :	Computation time
T_c :	Clock period

T_s :	Average time to compute one sample by stage 1
$w(i)$:	Transformed wavelet coefficients
X :	Wordlength of input samples
Y :	Wordlength of filter coefficients
Z :	Number of layers of a MAC-cell network
λ :	Parameter representing the design complexity with respect to the synchronization of pipeline stages
η :	Parameter representing hardware utilization efficiency
$\psi(.)$:	Wavelet function
$\varphi(.)$:	Scaling function

Chapter 1

Introduction

1.1 Background

In recent years, the discrete wavelet transform (DWT) has been widely and increasingly used in many fields such as image compression, speech analysis and pattern recognition because of its capability of decomposing a signal at multiple resolution levels [1]–[18]. The DWT decomposes a signal into components in different octaves or frequency bands by choosing appropriate scaling and shifting factors where the small scaling factor corresponds to fine details of the signal and the large scaling factor to coarse details, and the shifting factor corresponds to the time or space localization of the signal [19]–[21]. In contrast to other transforms, such as Fourier or cosine transforms where the signals are represented in frequency domain only, the DWT decomposes a signal so that it is represented more efficiently and localized in both time (space) and frequency domains. In other words, in the DWT, the time (space) information is not lost in the transformed signal, which is very attractive for the analysis of signals, especially for signals with non-stationary or transitory characteristics [22], [23].

In accordance with the multiple-level decomposition of a signal, the computation of the DWT can be performed by repeating a process in which a fully scalable window is shifted along the dimensions of the signal with the window size becoming shorter in each repetition. The computing processes of the DWT can be carried out by executing recursively a set of instructions developed in software programs such as SimuWave in Simulink, Wavelet toolbox in MATLAB and WavBox in Toolsmiths [24]–[27]. The software implementation for the computation of the DWT is flexible in setting different values of the parameters of the transform and changing the codes for the algorithms. Regardless of the effort devoted to the design of software algorithms and optimized codes for their implementations, no general-purpose or DSP processor used for their implementation can provide a performance in terms of the computing speed and resource optimization that can possibly be achieved by a hardware implementation [28]–[33].

Hardware implementations, in which the computation of the DWT is performed by a custom hardware circuit, it is possible to address the requirements of specific applications such as the speed, power or size of the circuit. In the literature, there exist a number of design efforts on the development of architectures for the DWT computation that focus on such requirements of applications [34]–[41]. However, many applications of the DWT computation involve large-volume data such as image or video. The fact that the DWT is multiple resolution level operation adds even more to the vastness of the data to be processed, which adversely affects the requirements of speed, power and the circuit area of the architectures for such applications. Thus, it remains a challenging task to design high-speed, low-power and area-efficient VLSI architectures to implement the DWT computation for real-time applications.

1.2 Motivation

In the past, several types of architectures have been proposed aimed at providing high-speed computation of the DWT using resource-efficient hardware. The architectures in [42]–[51] employ a single processor to perform the computations of all the resolution levels of the DWT, mostly based on the recursive pyramid algorithm (RPA) [52]. Naturally, by using a single processor in these architectures, the computations of the various resolution levels of the DWT are performed in a sequential manner, since the computation at one resolution level requires the output data from its preceding level. Even though these architectures have low design and hardware complexities, they do not focus on providing a fast computation of the DWT. Therefore, this type of architectures is not attractive for real-time applications. In an effort to overcome the problem of slow computation, architectures that employ two or more parallel processors have been proposed [53]–[60]. In this type of architectures, the computation associated with one level is performed by more than one processor thereby increasing the overall processing speed of the DWT computation. These type architectures even though provide parallelism to the computations associated with a given resolution level, they do not have parallelism between the resolution levels. In order to further improve the parallelism for the DWT computation, and hence the computational speed, the architectures that employ a number of pipelined stages, each performing the task of one or more resolution levels of the DWT, have been proposed [61]–[67]. The focus in the design of these architectures is on introducing some parallelism in the computations associated with the multiple resolution levels of the DWT, thus aiming at providing a high throughput and overall a short computing time.

From the foregoing discussion, it is clear that pipeline architectures are well suited for the DWT computation of large-volume data. However, no systematic approach seems to exist in determining the number of stages, mapping of the resolution levels to the stages and the design of the stages themselves so as to minimize the computation time and maximize the utilization of the hardware resource of the pipeline.

1.3 Scope of the Thesis

The operation of discrete wavelet transform has the characteristic of getting the amount of computations in successive resolution levels reduced by a factor of two along each dimension of the signal. This thesis in conformity with this inherent feature of the DWT undertakes a study of designing fast and hardware resource-efficient pipeline architectures for the computation of the 1-D and 2-D discrete wavelet transforms. With this overall objective, the mapping of the computational tasks associated with the various resolution levels of the DWT to an optimum number of pipeline stages is first investigated, and then an efficient design of the stages are explored from the standpoint of maximizing the inter-stage and intra-stage parallelisms of the pipeline architectures with an efficient utilization of the hardware resources employed.

1.4 Organization of the Thesis

The thesis is organized as follows.

In Chapter 2, first, discrete mathematical models for the computation of 1-D and 2-D wavelet transforms are presented and the methods for their computation are described.

The existing architectures for the computation of these transforms are then reviewed and classified.

In Chapter 3, a study for developing a scheme aimed for the design of a resource-efficient pipeline architecture for fast computation of the 1-D DWT is undertaken. With this goal in mind, the number of stages of the pipeline is first determined so as to map the computational tasks of the various resolution levels of the DWT to the pipeline stages in a most optimal manner. The second part of this chapter then focuses on the design of the pipeline stages themselves so as to maximize the inter-stage and intra-stage parallelisms. A case study for the design and FPGA implementation of a pipeline architecture is undertaken to illustrate and validate the proposed scheme and to compare it with other existing schemes for the design of architectures for the 1-D DWT computation.

Since the complexities in two-dimensional signal processing are generally quite different from that in 1-D case and they often call for a different approach for their solutions, in Chapter 4, an investigation is undertaken for developing a scheme for a pipeline architecture for the computation of the 2-D DWT. Various components of the design of pipeline architecture are looked into with the overall goal being the same as that in the design of the 1-D DWT pipeline architectures, namely, the development of a fast resource-efficient pipeline architecture. A circuit for a 2-D DWT computation is designed, simulated and implemented in FPGA, and the simulation and implementation results are then compared with those for the existing architectures to validate the efficiency of the proposed design.

Chapter 5 concludes the thesis by summarizing the work contained therein, highlighting the contributions made, and stating the scope of some possible future work arising from the work of this thesis.

Chapter 2

Background Material and Related Previous Work

This chapter provides background material necessary for the development of the architectures of the 1-D and 2-D discrete wavelet transforms undertaken in the following chapters. First, the mathematical formulations of the 1-D and 2-D discrete wavelet transforms are presented, and methods for their computations are described. This is followed by a review of the various existing architectures, classified as single-processor, parallel-processor and pipeline architectures for the 1-D and 2-D DWT computations.

2.1 Fundamentals of the Discrete Wavelet Transform

2.1.1 Definitions of Wavelet Transforms

The wavelet transform was first introduced by Jean Morlet in 1981 [68]. The continuous wavelet transform (CWT) of a signal $f(x) \in \mathbf{L}^2(R)$ (Hilbert space) is an integral operation defined as

$$w(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{+\infty} f(x) \psi(\frac{x-b}{a}) dx \tag{2.1}$$

where $w(a,b)$ are the wavelet coefficients, $\psi(\frac{x-b}{a})$ are the wavelets generated by a basic wavelet function $\psi(x) \in \mathbf{L}^2(R)$, the so-called the *mother wavelet*, a is the dilation parameter that scales the wavelet function by compressing or stretching it, and b is the translation parameter that locates the position of the wavelet function by shifting it. It is seen from this definition that the wavelet transform is a linear operation. By changing the variable $x = ax'$ and expressing the dilation parameter as $a = a_1^{v_j}$, where a_1 and v_j (j=1, 2, 3, …) are real numbers, (2.1) becomes

$$w(a_1^{v_j}, b) = \sqrt{a_1^{v_j}} \int_{-\infty}^{+\infty} f(a_1^{v_j} x') \psi(x' - \frac{b}{a_1^{v_j}}) dx' \tag{2.2}$$

Therefore, the wavelet transform can be seen as a decomposition of the signal $f(x)$ into a number of resolution levels with j = 1, 2, 3, ⋯ . Fig. 2.1 shows a hierarchical structure for the decomposition of a signal $f(x)$ into multiple resolution levels of the wavelet transform. It is seen from this figure that, in order to obtain the wavelet coefficients $w(a_1^{v_j}, b)$ of a certain resolution level j, the signal $f(x)$ is first scaled by a factor of $a_1^{v_j}$, and then integrated with a dilated and translated wavelet function $\psi(x - \frac{b}{a_1^{v_j}})$ followed by a multiplication by the magnitude factor $\sqrt{a_1^{v_j}}$. It is also seen from this figure that the wavelet transform is very suitable for analyzing the hierarchical structure of the function $f(x)$ because of its mathematical microscopic property that allows a signal to be represented by a number of functions with automatic scalability [69].

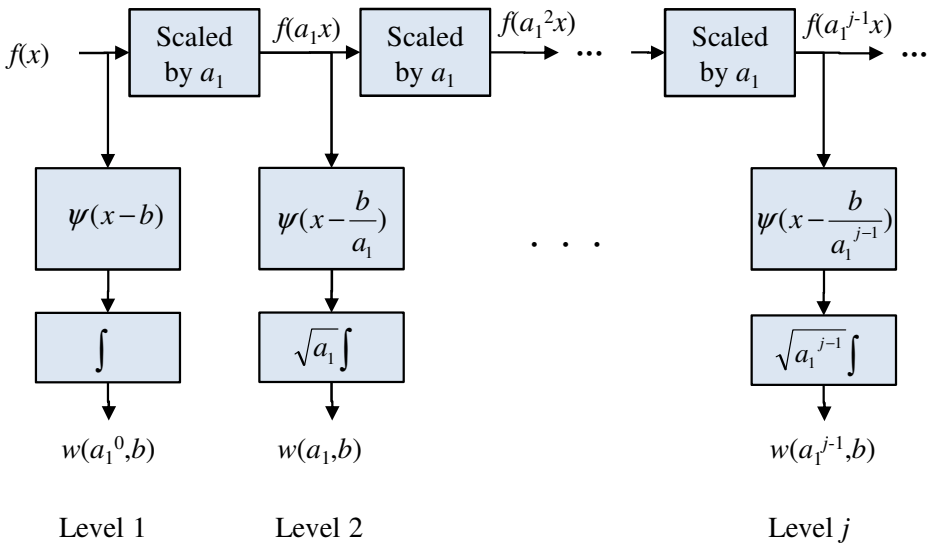


Figure 2.1: Hierarchical structure for the decomposition of a signal $f(x)$ into multiple resolution levels of the wavelet transform.

2.1.2 Mathematical Formulations

(a) Expression for the 1-D DWT

According to (2.1), continuous wavelet transform may use an infinite number of wavelets $\psi(\frac{x-b}{a})$. Thus, it is not practical in analysis of a signal due to the redundant calculation resulting from the dilation and translation parameters [70]. Discrete wavelets are introduced to address this problem. Discrete wavelets are not continuously scalable and translatable, but can be scaled and translated in discrete steps as denoted by

$$\psi_{j,k}(x) = a_0^{j/2}\psi\left(a_0^{\ j}x - kb_0\right) \tag{2.3}$$

where a_0 and b_0 are scale and translation factors, respectively, and k and the scale index j are two integers. Generally, the value of the scale factor a_0 is chosen as two so as to achieve a dyadic sampling along the frequency axis, and the translation factor b_0 has a

value of unity so as to achieve a dyadic sampling along the time axis. Note that the function $\psi_{j,k}(x)$ has fine scale or high frequency when the scale index *j* becomes large. Discrete wavelets are still continuous functions of *x* but discretized in the time-scale space. The discrete form of the wavelet transform can now be formulated as

$$w(j,k) = \int_{-\infty}^{+\infty} f(x)\psi_{j,k}(x)dx \tag{2.4}$$

A dilation of wavelet functions by a factor of two in the time domain leads to a reduction of their frequency by one-half. Since the wavelet functions have a feature of a band-pass filter, in order to cover the entire frequency band down to zero when decomposing a signal, an infinite number of levels would be required. To solve this problem, a scaling function $\phi(x)$, also called the *father wavelet*, was introduced by Mallat in 1989 [71]. The scaling function $\phi(x)$ has a feature of a lowpass filter, and must satisfy the two-scale dilation property given by

$$\phi(x) = \sqrt{2}\sum_{k} h(k)\phi(2x-k) \tag{2.5}$$

$$\psi(x) = \sqrt{2}\sum_{k} g(k)\phi(2x-k) \tag{2.6}$$

where $h(k)$ and $g(k)$ are the coefficients of two digital filters. If functions $\phi_{j,k}(x) = 2^{j/2}\phi\left(2^{j}x-k\right)$ and $\psi_{j,k}$ are orthogonal, the coefficients $h(k)$ and $g(k)$ are the inner products $\langle \phi_{0,0}, \phi_{-1,k} \rangle$ and $\langle \psi_{0,0}, \phi_{-1,k} \rangle$, respectively.

Since the scaling function has the feature of a lowpass filter, it sets a low bound on frequency for the decomposition of a signal. For the decomposition of any given scale index *j*, the scaling functions $\phi_{j,k}$ and wavelet functions $\psi_{j,k}$ share the entire frequency band, and only the frequency band of the scaling functions $\phi_{j,k}$ will be covered by further

decompositions of the scale index $j-1$, as shown in Fig. 2.2 [72]. Therefore, by combining the scaling and wavelet functions, the entire frequency band is covered with a limited number of resolution levels.

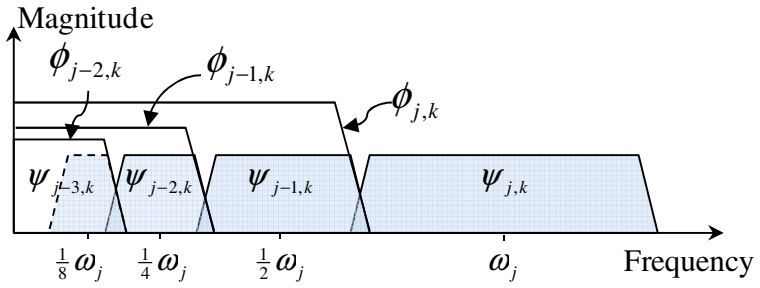


Figure 2.2: Frequency bands covered by the scaling and wavelet functions.

Due to the dilation property of the scaling and wavelet functions and their relations given by (2.5) and (2.6), for any scale index j, a signal $f(x)$ can always be expanded in terms of the scaled and translated wavelet and scaling functions, $\psi(2^j x-k)$ and $\phi(2^j x-k)$, as

$$f(x)=\sum_k c(j,k)\phi(2^j x-k)+\sum_k w(j,k)\psi(2^j x-k) \tag{2.7}$$

where $c(j,k)$ and $w(j,k)$ are, respectively, the scaling and wavelet coefficients associated with a scale index j. In order to obtain the coefficients $c(j,k)$ and $w(j,k)$, we need to compute the inner products $\langle f(x),\phi(2^j x-k)\rangle$ and $\langle f(x),\psi(2^j x-k)\rangle$, respectively. Using (2.4)–(2.7) and after some manipulation, we can obtain the coefficients, $c(j,k)$ and $w(j,k)$, as

$$c(j,k)=\sum_m h(m-2k)c(j+1,m) \tag{2.8}$$

$$w(j,k)=\sum_m g(m-2k)c(j+1,m) \tag{2.9}$$

It is seen from the above two equations that the wavelet or scaling coefficients at a resolution level with the scale index *j* are formulated as a convolution of the coefficients of a digital filter and the scaling coefficients at the resolution level with the scale index (*j*+1).

(b) Expression for the 2-D DWT

In order to decompose a 2-D signal, the 1-D scaling and wavelet functions have to be extended to two dimensions. A 2-D function can be obtained simply by multiplying two 1-D functions along *x* and *y* directions, respectively. Thus, the 2-D scaling function can be generated from the 1-D scaling functions, as

$$\phi(x, y) = \phi(x)\phi(y) \tag{2.10}$$

Using (2.10) in the manner similar to that for obtaining (2.8), the 2-D scaling coefficients associated with the scale index *j* for the 2-D DWT can be obtained as

$$c(j, k_x, k_y) = \sum_{m_x} \sum_{m_y} h(m_y - 2k_y) h(m_x - 2k_x) c(j+1, m_x, m_y) \tag{2.11}$$

Similar to (2.10), three types of 2-D wavelet functions, namely, vertical wavelet $\psi^{(v)}(x)$, horizontal wavelet $\psi^{(h)}(x)$, and diagonal wavelet $\psi^{(d)}(x)$, can be obtained using the 1-D scaling and wavelet functions as

$$\psi^{(v)}(x, y) = \phi(x)\psi(y) \tag{2.12}$$

$$\psi^{(h)}(x, y) = \psi(x)\phi(y) \tag{2.13}$$

$$\psi^{(d)}(x, y) = \psi(x)\psi(y) \tag{2.14}$$

which lead to three types of wavelet coefficients given by

$$w^{(v)}(j, k_x, k_y) = \sum_{m_x} \sum_{m_y} g(m_y - 2k_y) h(m_x - 2k_x) c(j+1, m_x, m_y) \tag{2.15}$$

$$w^{(h)}(j,k_x,k_y)=\sum_{m_x}\sum_{m_y}h(m_y-2k_y)g(m_x-2k_x)c(j+1,m_x,m_y) \tag{2.16}$$

$$w^{(d)}(j,k_x,k_y)=\sum_{m_x}\sum_{m_y}g(m_y-2k_y)g(m_x-2k_x)c(j+1,m_x,m_y) \tag{2.17}$$

It is seen from (2.11) and (2.15)–(2.17) that four components of the 2-D DWT at the resolution level with a scale index j are produced using the scaling coefficients at the level with a scale index (j+1). It should be noted that if the 2-D scaling and wavelet functions used in the 2-D DWT are non-separable in terms of x and y, the product of two 1-D filter coefficients in the right side of (2.11) and (2.15)–(2.17) will be replaced by a 2-D filter coefficient.

2.1.3 Computations of Discrete Wavelet Transforms

(a) Computation of the 1-D DWT

According to (2.8) and (2.9), the method for computing the 1-D DWT can be viewed as a sequence of operations along a binary tree consisting of a set of two-channel filter banks [73]. Fig. 2.3 shows an example of a binary tree for a 3-level DWT computation of 1-D signal $s(n)$. It is seen from this figure that the decomposition at any level of the DWT is computed by using a two-channel filter bank consisting of one highpass filter $G_{\mathrm{H}}(z)$ and one lowpass filter $G_{\mathrm{L}}(z)$, followed by a decimation operation by a factor of two in each channel. For a given resolution level j (j=1, 2, 3), the output samples of the two channels consist of a lowpass component $c_j^{(\mathrm{L})}(n)$ and a highpass component $w_j^{(\mathrm{H})}(n)$, of which only the component $c_j^{(\mathrm{L})}(n)$ is used as input for the decomposition at the next level j+1. The computation for the resolution level j has a complexity of $\mathrm{O}(N_0L/2^{j-1})$, where N_0 and L are, respectively, the number of samples of the input signal $s(n)$ and the length of

each of the two filters. It should be noted that as the resolution level j increases, the dilation parameter of the wavelets associated with the resolution level j becomes smaller and smaller, which results in representing the signal by functions having a coarser scale.

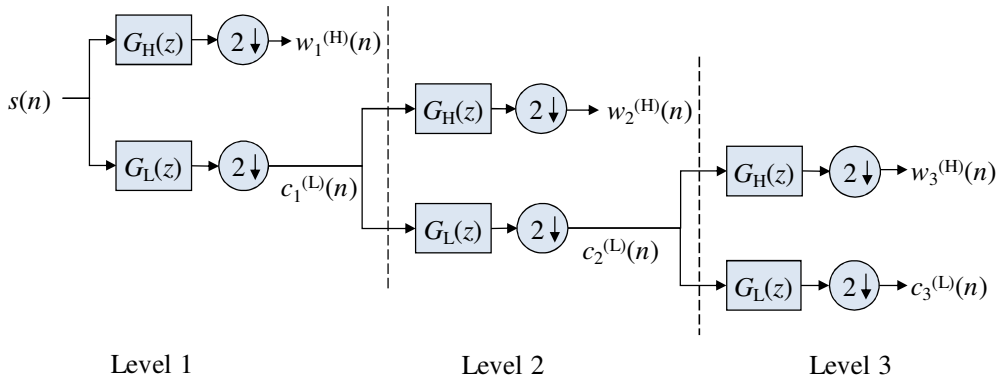


Figure 2.3: Binary tree representation of a 3-level 1-D DWT decomposition.

(b) Computation of the 2-D DWT

The computation of the 2-D DWT is more involved than that of the 1-D DWT, both in terms of the amount of processing as well as the complexity of the algorithm used for the computation.

(i) Separable Approach for the 2-D DWT Computation

A straightforward way to perform the computation of the 2-D DWT is to use a separable approach. In the separable approach, the impulse response $G(z_1, z_2)$ of each 2-D filter used for the DWT computation is product separable, i.e., $G(z_1, z_2) = G_1(z_1)G_2(z_2)$, The filter $G_1(z_1)$ is used to process the 2-D data of successive rows (columns). Then, the resulting 2-D data is processed successively along the columns (rows) using the filter $G_2(z_2)$. A binary tree representation for a 2-level DWT computation of 2-D signal $s(n_1, n_2)$ based on the separable approach is shown in Fig. 2.4. It is seen from this figure that the

computation for the decomposition of a given level j consists of two decomposition steps: row-wise decomposition of the 2-D input data using and column-wise decomposition of the 2-D data resulting from the row-wise decomposition. In the row-wise decomposition, each row of the 2-D input data is filtered using the two-channel horizontal filter bank ($G_{1H}(z_1)$ or $G_{1L}(z_1)$) and then downsampled by a factor of two, to produce horizontal highpass and lowpass components, each component having one-half of the numbers of samples in the rows of the 2-D input data. In the column-wise decomposition, each column of the two resulting components is filtered by using the two-channel vertical filter bank ($G_{2H}(z_2)$ or $G_{2L}(z_2)$) and downsampled by a factor of two so that in total four components, specified as the HH component $w_j^{(HH)}(n_1, n_2)$, LH component $w_j^{(LH)}(n_1, n_2)$, HL component $w_j^{(HL)}(n_1, n_2)$ and LL component $c_j^{(LL)}(n_1, n_2)$, are obtained as outputs of the given level j. Among the four outputs, only the LL component $c_j^{(LL)}(n_1, n_2)$ is used for the computation of the next resolution level, which is an iteration of the above two steps.

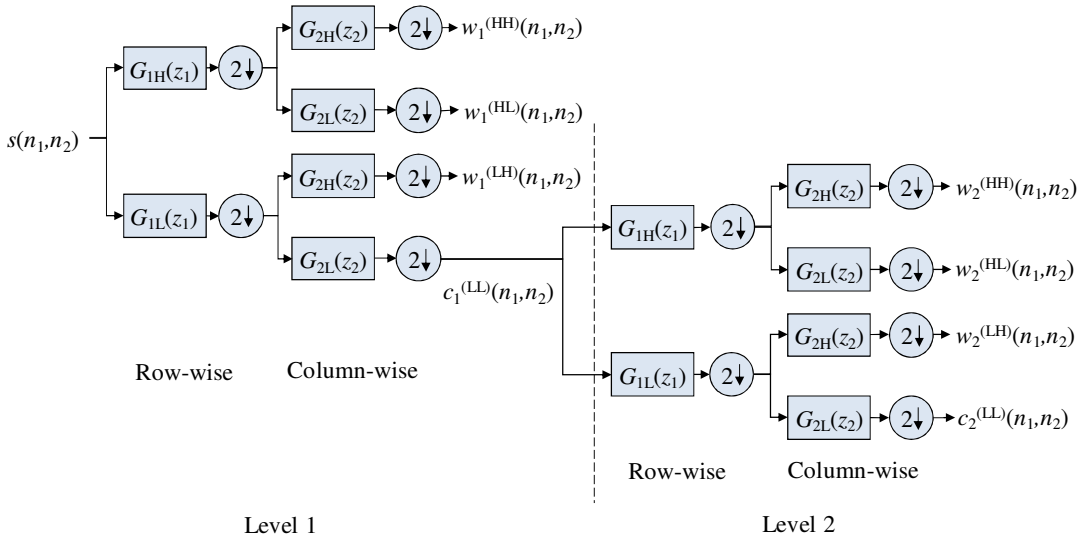


Figure 2.4: Binary tree representation of the computation of a 2-level 2-D DWT based on separable approach.

It should be noted that each of the four resulting components has one-quarter of the number of samples of the 2-D input data to the jth level. The computation of the resolution level j has a complexity of $O(N_0M_0L/4^{j-1})$, where N_0 and M_0 are the numbers of the rows and columns of the 2-D input data.

(ii) Non-separable Approach for the 2-D DWT Computation

Obviously, separable approach is a simple way to compute the 2-D DWT. However, separable filters being a special class of 2-D filters are not capable to approximate well all arbitrary frequency responses. In this regard, a non-separable approach of the 2-D computation provides more flexibility. In the non-separable approach depicted in Fig. 2.5, the DWT of a 2-D signal $s(n_1, n_2)$ is computed by carrying out four separate 2-D filtering operations using four 2-D filters: a highpass-highpass (HH) filter $G_{HH}(z_1, z_2)$, a highpass-lowpass (HL) filter $G_{HL}(z_1, z_2)$, a lowpass-highpass (LH) filter $G_{LH}(z_1, z_2)$, and a lowpass-lowpass (LL) filter $G_{LL}(z_1, z_2)$. The output signals of these four filters are then

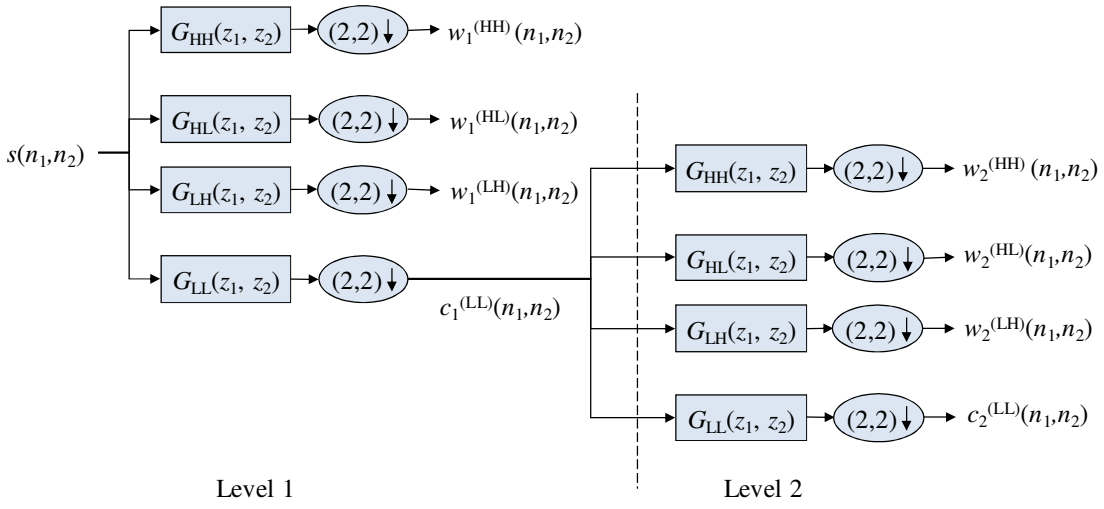


Figure 2.5: Representation of the computation of a 2-level 2-D DWT based on non-separable approach.

decimated by a factor of two in the horizontal and vertical directions producing, respectively, the HH, HL, LH and LL components. The computation of the resolution level j using the non-separable approach has a complexity of $O(N_0M_0L^2/4^{j-1})$, where N_0 and M_0 are, respectively, the numbers of rows and columns of the 2-D input data, and L^2 is the number of coefficients in each of the $L\times L$ 2-D filters.

2.2 Review of the Architectures

2.2.1 Categorization of the Architectures

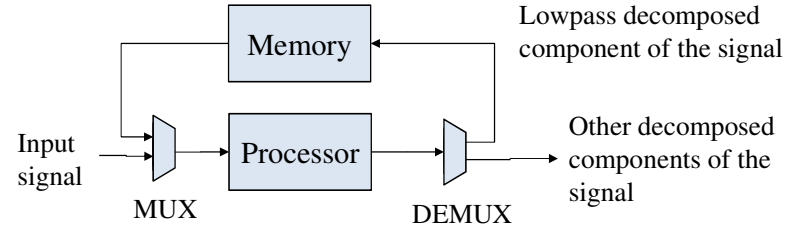
In recent years, many architectures have been proposed for the DWT computation [74]–[109]. These architectures aim at providing high performances, in terms of their speed, area, throughput, latency and power consumption. The filtering operation involved in the DWT computation is usually the convolution operation, that is, FIR filtering. The structure of the filter could be a direct realization, or it could be a systolic, lattice, bit-wise or lifting based realization depending on the way that the basic convolution operation is manipulated or formulated [110]–[128]. For example, the lifting scheme proposed by Sweldens [129], [130] exploits the relationship that exists between the lowpass filter G_L and the highpass filter G_H for the computation of the DWT. In this case, the polyphase matrix $\mathbf{Q(z)} = [G_L\ G_H]^T$ can then be factorized as

$$\mathbf{Q(z)} = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \prod_{i=1}^{m} \begin{bmatrix} 1 & 0 \\ t_i(z) & 1 \end{bmatrix} \begin{bmatrix} s_i(z) & 1 \\ 1 & 0 \end{bmatrix} \tag{2.18}$$

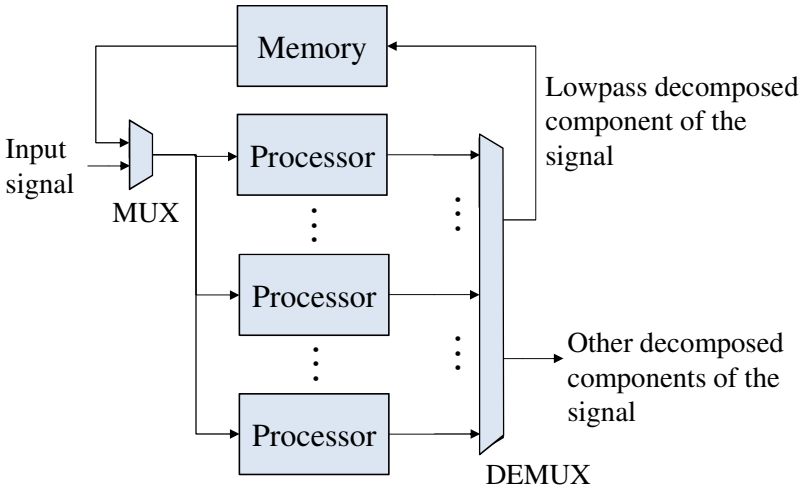
where K is a constant, and the two so called Laurent polynomials $s_i(z)$ and $t_i(z)$ have low orders. It is seen from (2.18) that the lifting-scheme based filtering operation requires a

cascade of lifting steps, and thus, leads to a large latency and a long critical path of the resulting lifting architecture.

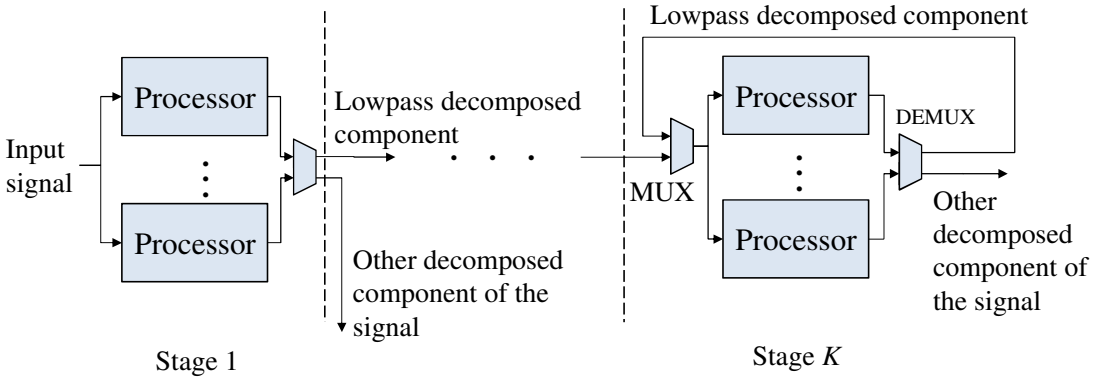
The filtering operation is carried out by using a processor that employs a certain type of filter structure. An architecture may use one or multiple such processors to perform the DWT computation. For the purpose of reviewing these existing architectures, we categorize them as single-processor architectures, parallel-processor architectures and pipeline architectures, depending on their configuration and the number of processors used by them. In a single-processor architecture, only one processor carries out the filtering operation by computing the samples of the DWT in a recursive manner. In a parallel-processor architecture, multiple processors are used to carry out the filtering operations so that more than one sample is computed at a time. In this type of architecture, the filtering operations to decompose the input signal into various components are carried out in parallel, whereas the computations of various resolution levels are still performed recursively by the parallel processors. In a pipeline architecture, a certain number of stages, each consisting of one or more processors, are pipelined so that the computation of each decomposition level as well as that of the multiple resolution levels are performed in parallel. Fig. 2.6 depicts the block diagrams for the three categories of the architectures. In each of these three broad categories, architectures may differ considerably because of the internal structures of processors employed for the filtering operation. In the following, examples of 1-D and 2-D architectures are given for each of the categories, and their salient features discussed.



(a)



(b)



(c)

Figure 2.6: Block diagrams of three types of architectures. (a) Single-processor architecture, (b) parallel-processor architecture, and (c) pipeline architecture.

2.2.2 Architectures for 1-D DWT Computation

(a) Single-processor Architectures

In the single-processor category, the VLSI architecture proposed by Guo *et al*. [42] is an example, in which only a single multiplier and a single adder, as shown in Fig. 2.7, are used, and thus, requires substantially large computation time. Fig. 2.8 depicts another example of this category, in which the processor utilizes a systolic array of multiply-accumulate (MAC) cells and a bank of shift registers for the filtering operations [43]. However, this architecture is slow because of the delays involved in the propagation of the signal through the array of MAC cells. The lifting-scheme based processor, proposed by Liao *et al*. [44], is yet another example of the single-processor architecture, in which the processor consists of a cascade of lifting steps, as shown in Fig 2.9, and it is used to compute the samples of the first resolution level at every other clock cycles and those of the other levels at the intervening clock cycles. However, a cascade of many lifting steps would result in quite a long critical path for this type of architecture.

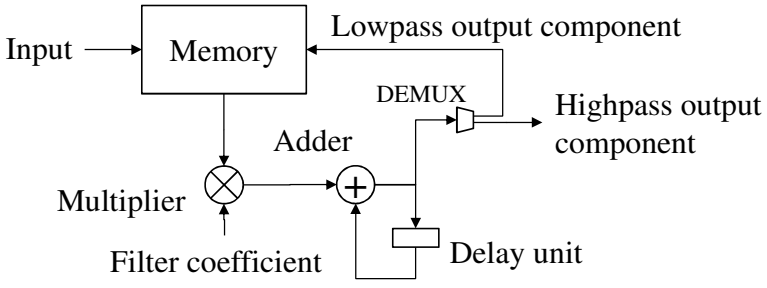


Figure 2.7: An architecture using one multiplier and one adder [42].

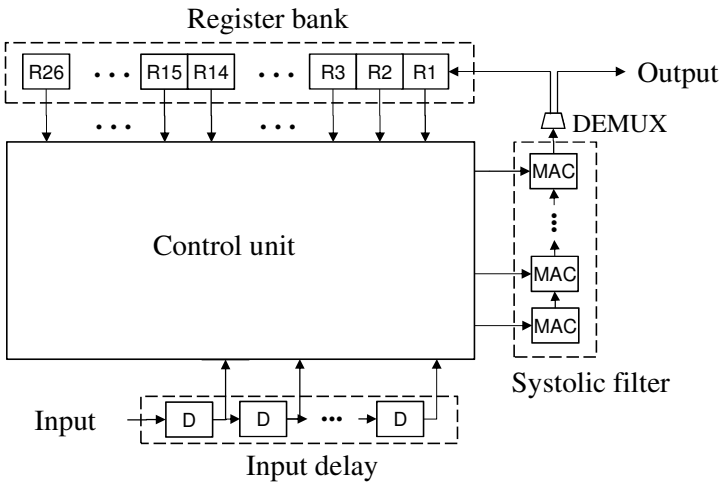


Figure 2.8: An architecture using a processor employing a systolic array of MAC cells [43].

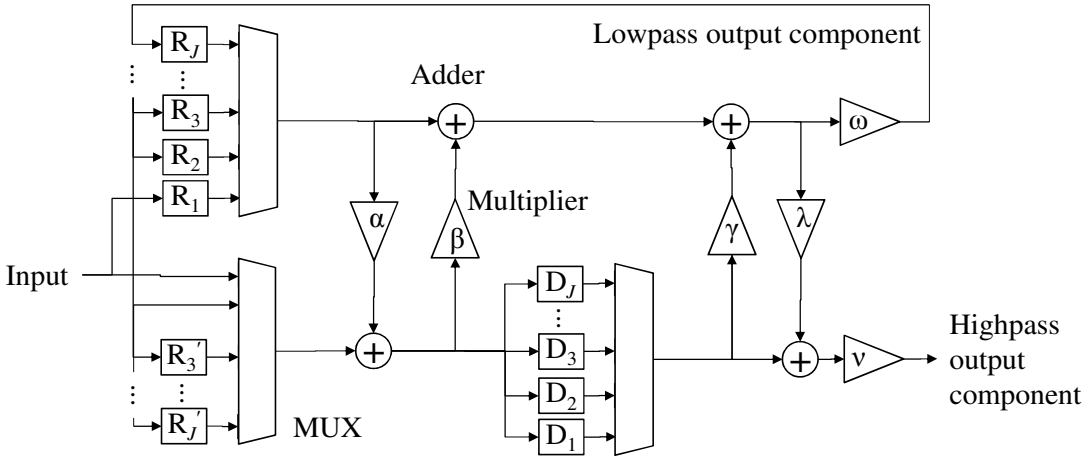


Figure 2.9: A lifting-based architecture using Daub-4 filters. R_j and D_j represent, respectively, the registers and delay units for the computation of the *j*th level [44].

(b) Parallel-processor Architectures

In the parallel-processor category, Chakrabarti and Vishwanath [53] have proposed an architecture that uses two processors operating in parallel, one for lowpass and the other for highpass filtering operations, and one storage unit, as shown in Fig. 2.10. Since

this architecture has as many memory blocks as the number of resolution levels for storing the lowpass data, it requires a large memory space.

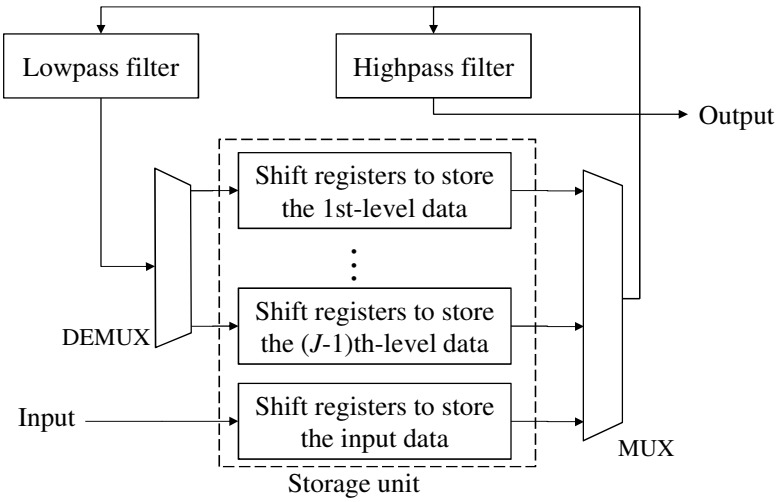


Figure 2.10: A parallel architecture proposed by Chakrabarti and Vishwanath [53].

The folded architecture proposed by Parhi and Nishitani [54] is an example of a parallel-processor architecture, in which a pair of lowpass and highpass systolic filters and a set of shift registers are used to perform the computations of multiple resolution

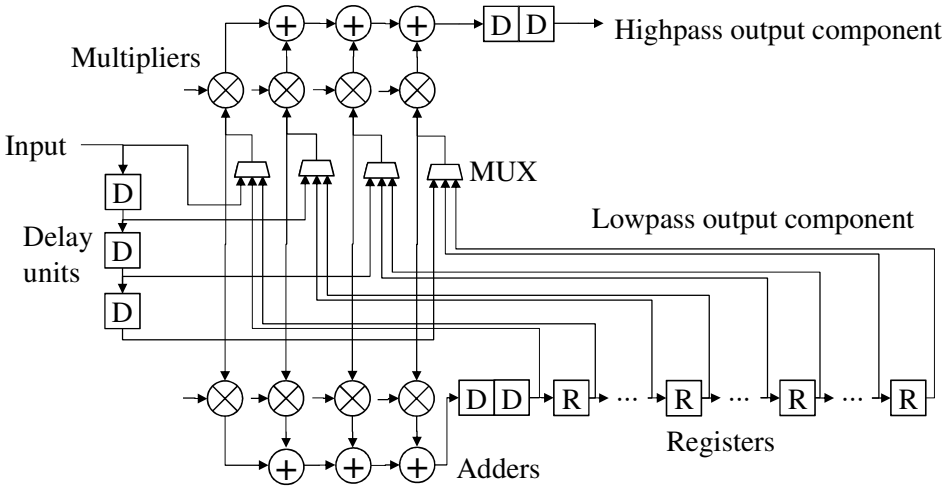


Figure 2.11: A folded architecture proposed by Parhi and Nishitani [54] using 4-tap filter.

levels, as shown in Fig. 2.11. This architecture requires complex routing, and has a low throughput rate for large-size filters.

Masud and McCanny [55] have proposed a two-processor architecture, as shown in Fig. 2.12, using *L*-tap lowpass and highpass filters operating in parallel. However, in this architecture, each of the two filters uses only *L*/2 MAC cells that operate on odd and even numbered coefficients in consecutive clock cycles. The architecture results in a large computation time and has a complex control unit.

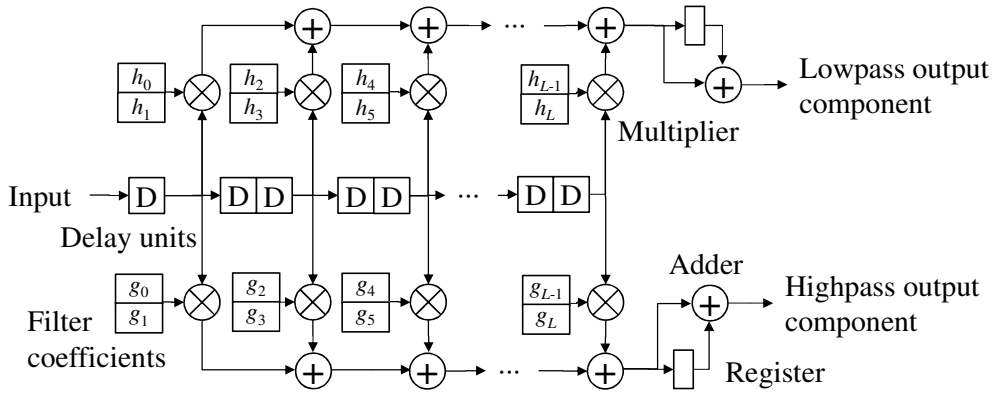


Figure 2.12: An architecture proposed by Masud and McCanny [55].

(c) Pipeline Architectures

The architecture proposed by Marino *et al*. [61] and shown in Fig. 2.13 is an example of a pipeline architecture, in which a number of pipeline stages B_j (j=1, 2, …, J) are employed for the computations of J resolution levels. The computation of the jth resolution level is performed by the jth stage using $W_j = \left| L/2^{j-2} \right|$ MAC cells, as shown in Fig. 2.13. However, the architecture requires a large amount of hardware resource when the number of resolution levels becomes large. Moreover, since the organization of MAC cells differs from stage to stage, the design complexity is quite high.

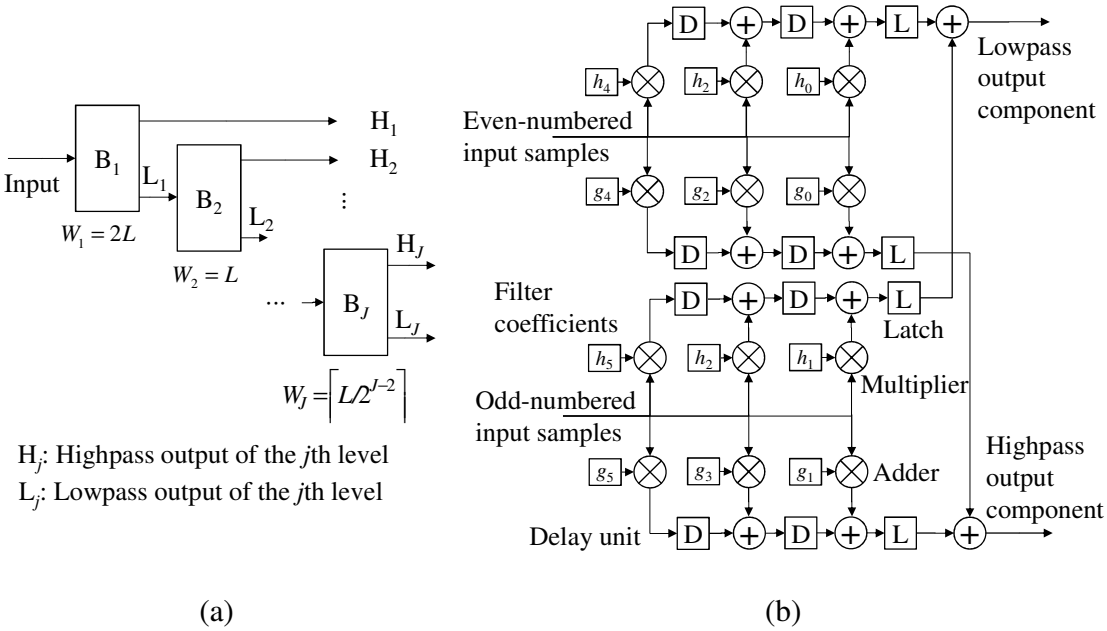
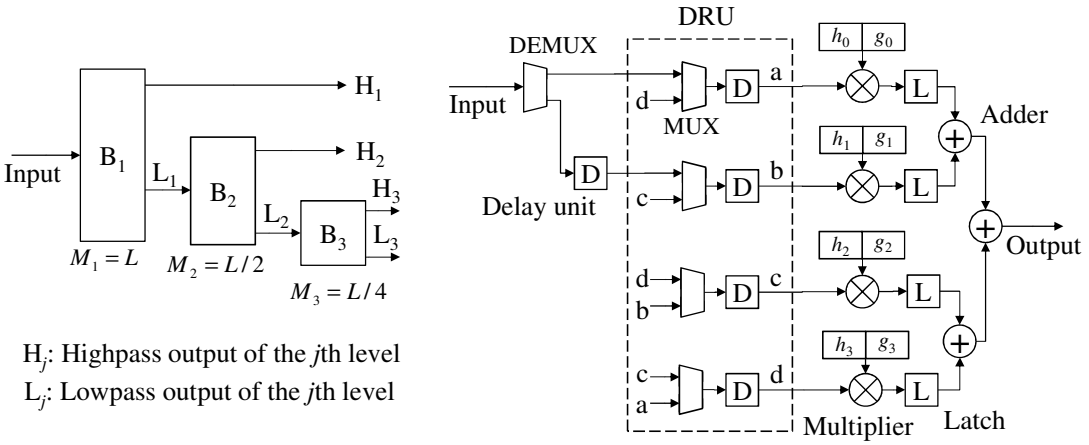


Figure 2.13: A pipeline architecture proposed by Marino *et al*. [61]. (a) Block diagram of the architecture. (b) Structure of B1 when a 6-tap filter is used.

Park [62] has proposed a scalable pipeline architecture, in which a certain stage B_j of the pipeline utilizes the number of parallel multipliers M_j that is one-half of that of the preceding stage, and uses a data recorder unit (DRU) for constructing the input data sequence, as shown in Fig. 2.14. However, the architecture is restricted for of the computation of the DWT which has three resolution levels and the number of filter length L is divisible by 4.

A lifting-scheme based architecture proposed by Chen [63] and shown in Fig. 2.15 is another example of a pipeline architecture, in which a certain number of identical stages, each consisting of splitting, predicting and updating units for the computation of a resolution level, are employed. In this architecture, since the number of computations performed by each stage is not consistent with the amount of hardware resource it employs, the architecture has low utilization of hardware resources.



(a) (b)

Figure 2.14: A scalable 3-stage architecture proposed by Park [62]. (a) Block diagram of the architecture. (b) Structure of the first stage.

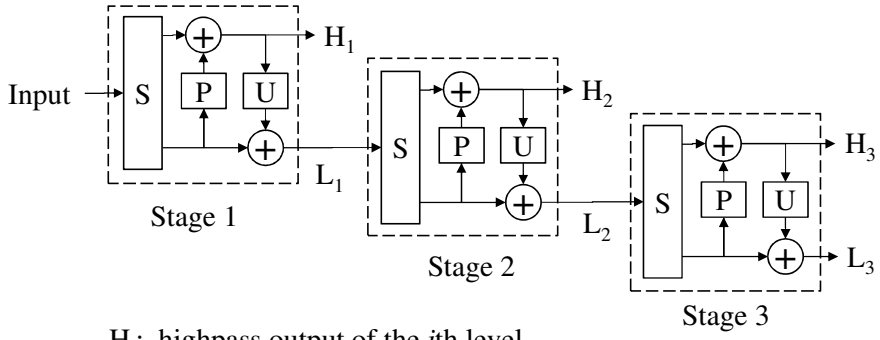


Figure 2.15: A lifting-scheme based pipeline architecture [63].

2.2.3 Architectures for 2-D DWT Computation

(a) Single-processor Architectures

In a single-processor category, the architecture of [47] is an example of a multiplierless architecture, and therefore, is restricted to only certain types of wavelets. Moreover, the architecture is not scalable. The architecture proposed by Movva and Srivivasan [48] is another example in the single-processor category for the 2-D DWT computation that uses the separable approach. In this architecture, the 2-D DWT is obtained by performing a row-wise computation followed by a column-wise computation using a single lifting-scheme based processor. The architecture is a low-speed and requires a large memory space. Hung *et al.* [49] have proposed a single-processor architecture using the non-separable approach, illustrated Fig. 2.16, in which an $L \times L$-tap filtering operation is carried out by a processor consisting of a cascade of three blocks: parallel multipliers, L accumulators along the row direction and one accumulator along the column direction. The architecture has low computational speed, since the samples of the four decomposed components are computed sequentially. The architecture [50] shown in Fig. 2.17 is another example of a single-processor architecture, in which the processor consists of $\lceil L/2 \rceil$ adders and $\lceil L/2 \rceil$ parallel processing blocks, where L is the filter length, followed by an accumulator. The architecture requires large storage (delay units) to store the lowpass-lowpass output components of various resolution levels. Fig. 2.18 shows another single-processor architecture proposed by Meher *et al*. [51] for the computation of the 2-D DWT using separable approach, in which the computation is performed by two blocks, referred to as Subcell-1 and Subcell-2. Subcell-1 employs parallel multiplication units and adders for row-wise filtering operation, whereas

Subcell-2 employs one delay cell and a systolic array of multiplication and adder units for column-wise operation.

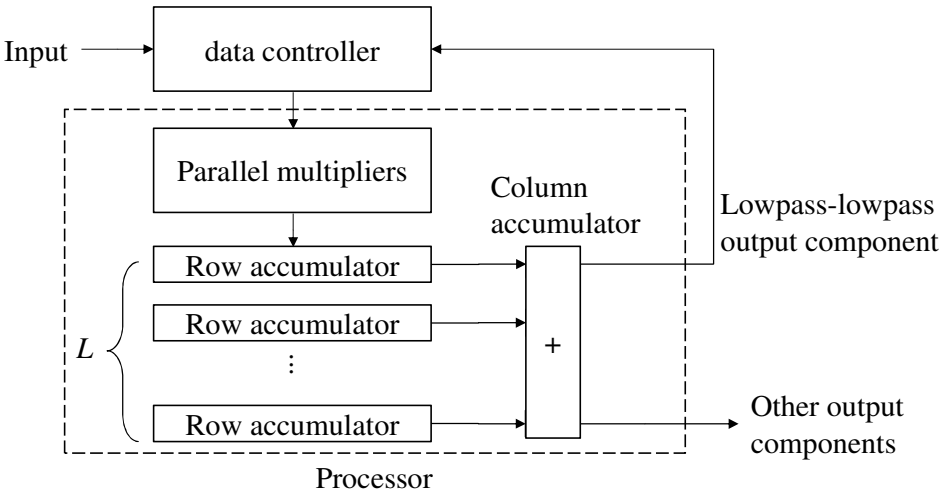


Figure 2.16: A single-processor architecture for the 2-D DWT computation [49].

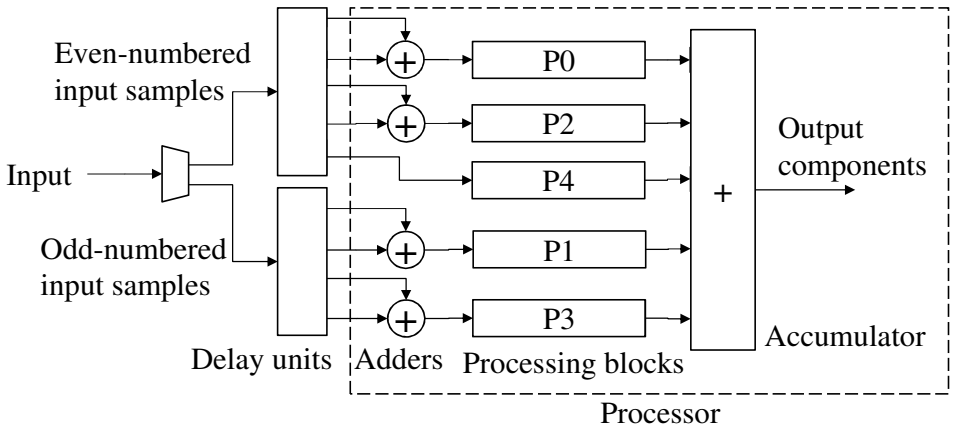


Figure 2.17: An architecture proposed by Uzun and Amira [50] for the 2-D DWT computation using 9/7-tap filters.

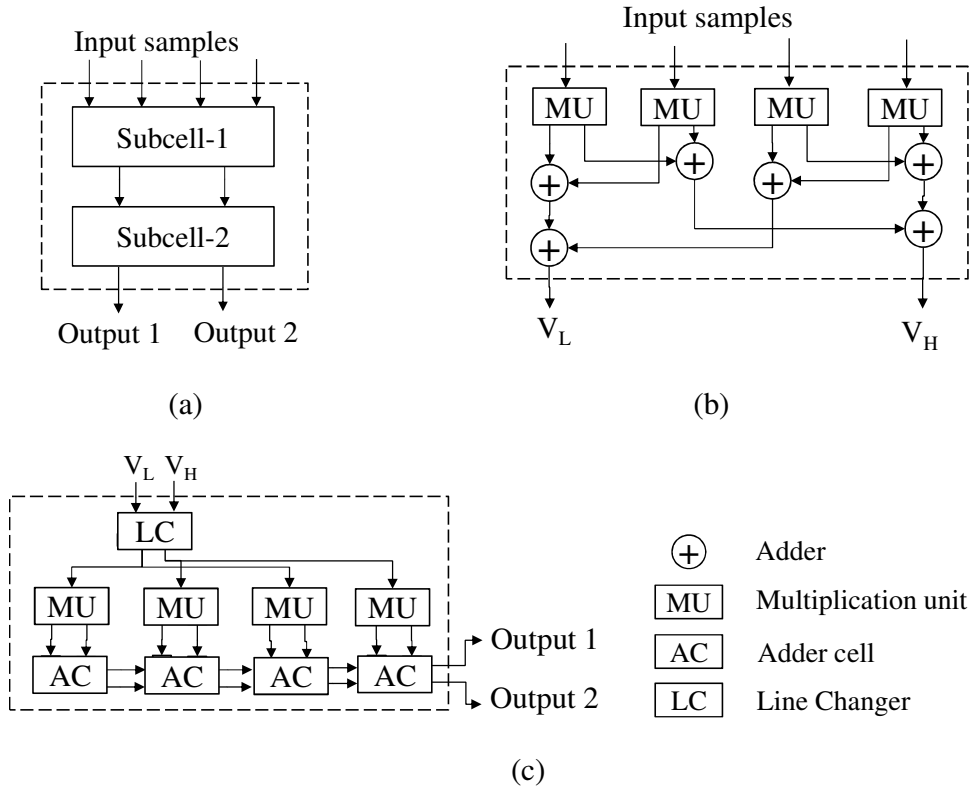


Figure 2.18: An architecture proposed by Meher *et al.* [51] for the 2-D DWT computation using separable approach. (a) Top-level architecture. (b) Structure of Subcell-1. (c) Structure of Subcell-2.

(b) Parallel-processor Architectures

In the category of parallel-processor architectures, Chakrabarti and Mumford [57] have proposed a four-processor architecture, as shown in Fig. 2.19, in which Filter Hor 1 performs the horizontal filtering operation of the resolution level $j = 1$, Filter Ver 1 and Filter Ver 2 perform, respectively, the vertical lowpass and highpass filtering operations of the resolution levels $j = 1, 2, 3, \ldots,$ and Filter Hor 2 performs the horizontal filtering operations of the resolution levels $j = 2, 3, \ldots.$ In this architecture, since the amounts of computations assigned to the four processors are not proportional to the amount of

hardware employed by them, the architecture has drawback of having low hardware utilization.

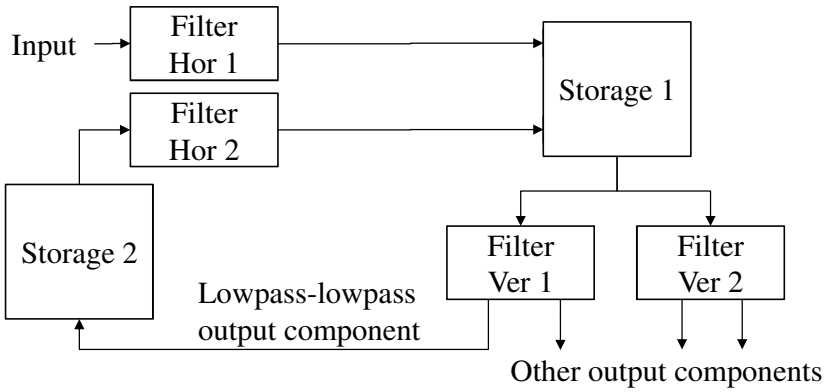


Figure 2.19: A 2-D DWT architecture proposed by Chakrabarti and Mumford [57].

The architecture [58] shown in Fig. 2.20 is another example of parallel-processor architecture for the 2-D DWT computation, in which two processors employing a poly-phase decomposition technique are used for row-wise filtering operation, and four other processors employing a filter coefficient folding technique are used for column-wise filtering operation. The architecture has a high design complexity, since the parallel processors have different structures.

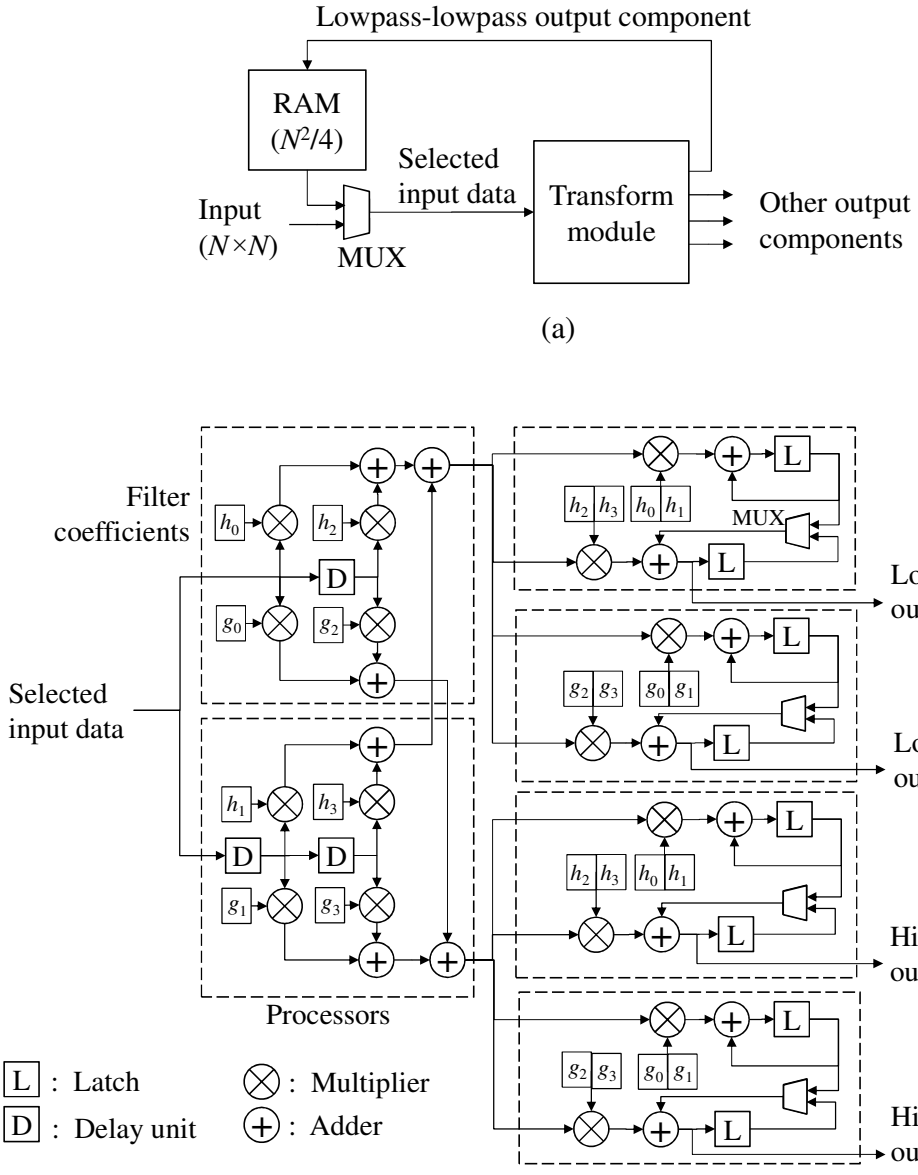


Figure 2.20: A parallel-processor architecture proposed by Wu and Chen [58] for the 2-D DWT computation. (a) Top-level architecture. (b) Structure of the transform module with six processors.

(c) Pipeline Architectures

The architecture [64] shown in Fig. 2.21 is an example of a pipeline architecture for the computation of the 2-D DWT. In this architecture, four processors, RF1, CF1, RF2 and CF2, form a pipeline, in which the first two processors are used to perform, respectively, the row-wise and column-wise operations of level 1 and remaining two to perform, respectively, the row-wise and column-wise operations of the remaining levels. The architecture has a large latency and requires large storage space.

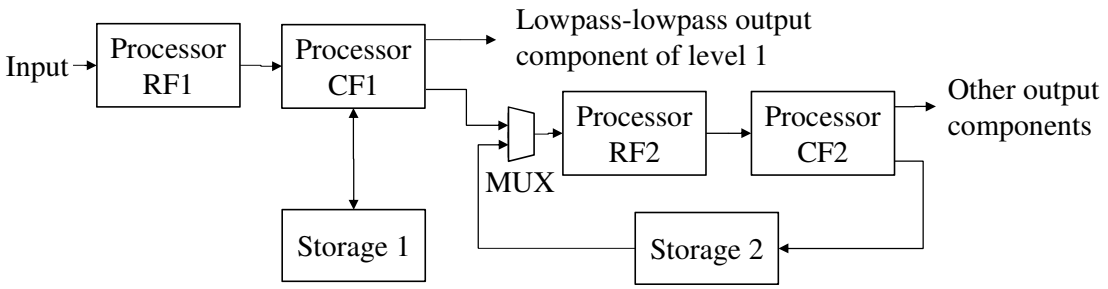


Figure 2.21: A pipeline architecture proposed by Jou *et al.* [64] for the 2-D DWT computation.

The architecture shown in Fig. 2.22 is another example of a pipeline architecture proposed by Mihić [65]. In this architecture, a J-level 2-D DWT is performed using a pipeline of $2J$ processors, each employing a semi-systolic array of MAC cells for row- or column-wise filtering operation of a resolution level. The architecture has a very large latency, and it is not practical for the computation of the DWT with large number of resolution levels.

The architecture of [66] (see Fig. 2.23) is yet another example of a pipeline architecture for the 2-D DWT computation. In this architecture, a pipeline of two stages, one for the computation of the first resolution level and the other for the computation of

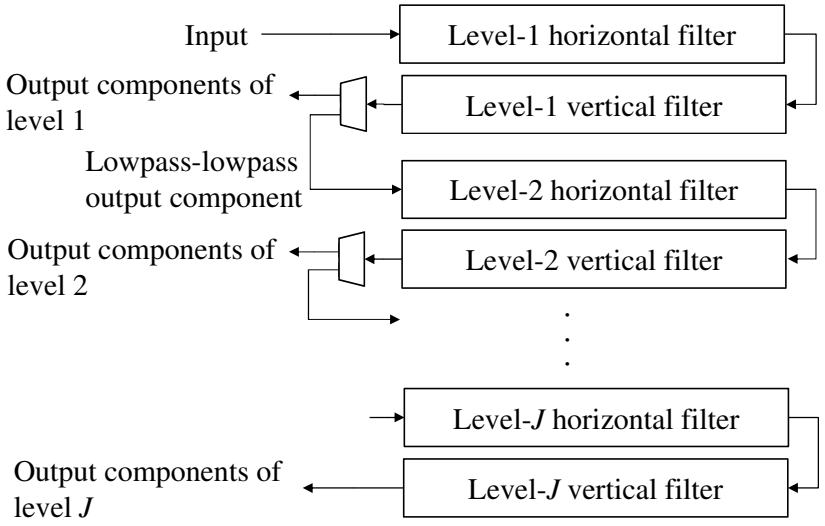


Figure 2.22: An architecture using a pipeline of 2*J* stages [65].

all the remaining levels, and each employing *L* parallel processing blocks, are used. The design complexity of this architecture is high, since the structures of the processing blocks are different. Also, it has a high hardware resource complexity, since each processing block has a large number of MAC cells.

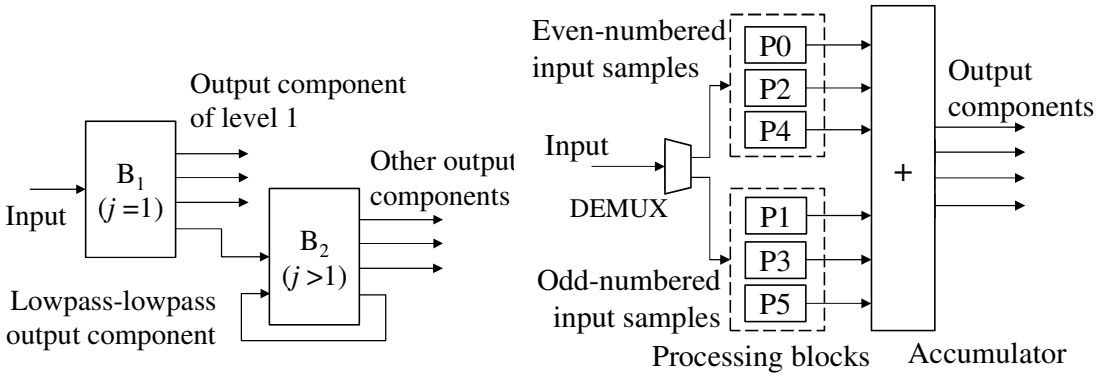


Figure 2.23: A two-stage pipeline architecture proposed by Marino [66]. (a) Top-level architecture. (b) The structure of the stage B_1 for L= 6.

2.3 Summary

In this chapter, starting with the mathematical definitions of the 1-D and 2-D wavelet transforms, discrete formulations have been provided for their practical computations. These discrete formulations are seen to follow a binary-tree structure for the computation of the 1-D DWT and, depending on the separable or non-separable approaches, a binary-tree or quadtree structures for the computation of the 2-D DWT. For the purpose of reviewing the existing architectures for the computation of the 1-D and 2-D wavelet transforms, they have been classified as single-processor, parallel-processor or pipeline architectures. A number of architectures from the literature in each of the categories, both for the 1-D and 2-D DWT computations, have been briefly reviewed. It has been seen that whereas the architectures in the single-processor and parallel-processor categories are efficient in terms of the speed and employment of hardware resources, respectively, the pipeline architectures are a good compromise between hardware resource complexity and speed.

Chapter 3

A Scheme for the Design of Pipeline Architectures for 1-D Discrete Wavelet Transform

In Chapter 2, a number of pipeline architectures [61]–[63] for the computation of the 1-D discrete wavelet transform were briefly reviewed. These architectures employ a large number of pipeline stages or utilize a large number of MAC cells to perform the filtering operations of the stages, and thus, have high complexity in terms of hardware resources [61], [62] or large latency [63]. In other words, the speed provided by these architectures is not commensurate with the hardware resources employed by them, The reason for these drawbacks is that the schemes used for the development of these architectures have not fully exploited certain characteristics inherent in the discrete wavelet transform.

In this chapter, a scheme for design of pipeline architectures for a fast computation of the DWT is proposed [131]–[133]. The goal of fast computation is achieved by minimizing the number and period of the clock cycles. The main idea in minimizing these two parameters is to optimally distribute the task of the DWT computation among the stages of the pipeline, and to maximize the inter- and intra-stage parallelisms of the

pipeline by synchronizing the operations of the stages optimally and by utilizing the available hardware resources judiciously.

The chapter is organized as follows. In Section 3.1, a matrix formulation for the 1-D DWT computation is presented. In Section 3.2, a study is undertaken to determine the number of stages in the pipeline to optimally assign to them the task of the 1-D DWT computation. Based on this study, in Section 3.3, a scheme for the design of a pipeline architecture is developed. In Section 3.4, the performance of the pipeline architecture for the DWT computation using the proposed design scheme is assessed and compared with that of other existing architectures. A specific example of designing an architecture for the DWT computation is also considered and the resulting architecture is simulated and implemented on an FPGA board in order to demonstrate the realizability and validity of the proposed scheme. Section 3.5 summarizes the work of this chapter by highlighting the salient features of the proposed design scheme and the resulting pipeline architectures.

3.1 Formulation of the 1-D DWT Computation

3.1.1 Matrix Formulation

The 1-D DWT of a signal is computed by performing the filtering operation repeatedly, first on the input data and then on the LL data after decimating it by a factor of two for the successive resolution levels. The filtering operation uses a quadrature mirror filter bank with lowpass and highpass filters to decompose the signal into lowpass and highpass subband signals, respectively. The transform can be expressed using a matrix formulation in order to provide a better insight into the underlining operations of

the DWT as well as to facilitate the proposed scheme for the design of the architecture for its computation.

Let the signal be denoted as $\mathbf{S}=[s_1, s_2, \cdots, s_{N-1}, s_N]^{\mathrm{T}}$, where N, the number of samples in the input signal, is chosen to be $2J$, J being an integer. Assume that h_i and g_i (i = 0,1,…,L−1) are the coefficients of the L-tap lowpass and highpass filters, respectively. Then, by expressing the transform matrices for the lowpass and highpass computations at the jth (j=1,2,…,J) level decomposition as

$$\mathbf{H}^{(j)}=\begin{bmatrix} h_0 & h_1 & h_2 & h_3 & \cdots & h_{L-1} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & \cdots & h_{L-3} & h_{L-2} & h_{L-1} & 0 & \cdots & 0 & 0 & 0 & 0 \\ & & & & & & \vdots & \vdots & & & & & & \\ 0 & 0 & 0 & 0 & & & \cdots & & & 0 & h_0 & h_1 & h_2 & h_3 \\ 0 & 0 & 0 & 0 & & & \cdots & & & 0 & 0 & 0 & h_0 & h_1 \end{bmatrix} \tag{3.1a}$$

$$\mathbf{G}^{(j)}=\begin{bmatrix} g_0 & g_1 & g_2 & g_3 & \cdots & g_{L-1} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & g_0 & g_1 & \cdots & g_{L-3} & g_{L-2} & g_{L-1} & 0 & \cdots & 0 & 0 & 0 & 0 \\ & & & & & & \vdots & \vdots & & & & & & \\ 0 & 0 & 0 & 0 & & & \cdots & & & 0 & g_0 & g_1 & g_2 & g_3 \\ 0 & 0 & 0 & 0 & & & \cdots & & & 0 & 0 & 0 & g_0 & g_1 \end{bmatrix} \tag{3.1b}$$

respectively, where both $\mathbf{H}^{(j)}$ and $\mathbf{G}^{(j)}$ have a size of $(N/2^j)\times(N/2^{j-1})$, the outputs of the transform at the jth level can be computed from the following:

$$\begin{bmatrix} \mathbf{C}^{(j)} \\ \mathbf{W}^{(j)} \end{bmatrix}=\begin{bmatrix} \mathbf{H}^{(j)} \\ \mathbf{G}^{(j)} \end{bmatrix}\cdot\mathbf{C}^{(j-1)} \tag{3.2}$$

where $\mathbf{C}^{(j)}$ and $\mathbf{W}^{(j)}$ represent the column vectors of size $N/2^j$ and consist of lowpass and highpass output samples, respectively, at the resolution level j, with $\mathbf{C}^{(0)}=\mathbf{S}$. It is clear from (3.1a) and (3.1b) that the lengths of the filters and the size of the input samples control the number of non-zero entries of the matrices involved, which in turn, determines the complexity of the DWT computation. If the decomposed signals are

required to be reassembled into the original form without loss of information, the lowpass and highpass filters must satisfy the perfect reconstruction condition given by

$$g_i = (-1)^{i+1} h_{L-1-i} \tag{3.3}$$

A border extension of the input signal becomes necessary for the processing of the samples on or near the border of a finite-length signal. There are generally three ways by which the border can be extended in a DWT computation, zero padding, symmetric padding and periodic padding [134]. Even though from the point of view of hardware cost, zero padding is the least expensive, the periodic padding is the most commonly used method for border extension, since it allows a precise recovery of the original signal at or near the border. This method extends the original sequence **S** by appending it with its first $L-2$ samples as

$$\mathbf{S_p} = [s_1, s_2, \cdots, s_{N-1}, s_N, s_1, \cdots, s_{L-3}, s_{L-2}]^{\mathrm{T}} \tag{3.4}$$

Thus, in order to operate on the padded input sequence $\mathbf{S_p}$, the transform matrices $\mathbf{H}^{(j)}$ and $\mathbf{G}^{(j)}$ have to be modified by appending each by additional $L-2$ columns. The elements of the appended columns in a row of a modified transform matrix assume a zero value, if all the filter coefficients already appear in the corresponding row of (3.1a) or (3.1b). Otherwise, the elements in the row are made to assume the missing values of the filter coefficients so that all the coefficients appear in that row of the modified transform matrix.

3.1.2 Reformulation of (3.2)

It is seen from (3.1) that due to the decimation-by-two requirement of the DWT, entries in the successive rows of matrices $\mathbf{H}^{(j)}$ and $\mathbf{G}^{(j)}$, and therefore, in their modified versions, are shifted to right by two positions. This property can be utilized to decompose the arithmetic operations in (3.2) into two parts so that the operations in one part can be performed simultaneously with those of the other one. For this purpose, we now decompose each of the modified transform matrices $\mathbf{H}^{(j)}$ and $\mathbf{G}^{(j)}$ by separating the even and odd numbered columns of each matrix into two sub-matrices. The resulting sub-matrices, taking into account the perfect reconstruction condition specified by (3.3), can be expressed as

$$\mathbf{H}_{even}^{(j)} = \begin{bmatrix} h_0 & h_2 & \cdots & h_{L-2} & 0 & 0 & & \cdots & & 0 \\ 0 & h_0 & \cdots & h_{L-4} & h_{L-2} & 0 & & \cdots & & 0 \\ & \vdots & & & & & & \vdots & & \\ 0 & & \cdots & & & 0 & h_0 & h_2 \cdots & h_{L-2} & 0 \\ 0 & & \cdots & & & 0 & 0 & h_0 \cdots & h_{L-4} & h_{L-2} \end{bmatrix} \tag{3.5a}$$

$$\mathbf{H}_{odd}^{(j)} = \begin{bmatrix} h_1 & h_3 & \cdots & h_{L-1} & 0 & 0 & & \cdots & & 0 \\ 0 & h_0 & \cdots & h_{L-3} & h_{L-1} & 0 & & \cdots & & 0 \\ & \vdots & & & & & & \vdots & & \\ 0 & & \cdots & & & 0 & h_1 & h_3 \cdots & h_{L-1} & 0 \\ 0 & & \cdots & & & 0 & 0 & h_1 \cdots & h_{L-3} & h_{L-1} \end{bmatrix} \tag{3.5b}$$

$$\mathbf{G}_{even}^{(j)} = -\begin{bmatrix} h_{L-1} & h_{L-3} & \cdots & h_1 & 0 & 0 & & \cdots & & 0 \\ 0 & h_{L-1} & \cdots & h_3 & h_1 & 0 & & \cdots & & 0 \\ & \vdots & & & & & & \vdots & & \\ 0 & & \cdots & & & 0 & h_{L-1} & h_{L-3} \cdots & h_1 & 0 \\ 0 & & \cdots & & & 0 & 0 & h_{L-1} \cdots & h_3 & h_1 \end{bmatrix} \tag{3.5c}$$

$$\mathbf{G}_{odd}^{(j)} = \begin{bmatrix} h_{L-2} & h_{L-4} & \cdots & h_0 & 0 & 0 & & & \cdots & 0 \\ 0 & h_{L-2} & \cdots & h_2 & h_0 & 0 & & & \cdots & 0 \\ & & \vdots & & & & & & \vdots & \\ 0 & & & \cdots & & 0 & h_{L-2} & h_{L-4} & \cdots h_0 & 0 \\ 0 & & & \cdots & & 0 & 0 & h_{L-2} & \cdots h_2 & h_0 \end{bmatrix} \tag{3.5d}$$

in which the entries in the successive rows are shifted to right by only one position. With this decomposition of the transform matrices, the DWT computation as given by (3.2) can be reformulated as

$$\begin{bmatrix} \mathbf{C}^{(j)} \\ \mathbf{W}^{(j)} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{even}^{(j)} \\ \mathbf{G}_{even}^{(j)} \end{bmatrix} \cdot \mathbf{C}_{even}^{(j-1)} + \begin{bmatrix} \mathbf{H}_{odd}^{(j)} \\ \mathbf{G}_{odd}^{(j)} \end{bmatrix} \cdot \mathbf{C}_{odd}^{(j-1)} \tag{3.6}$$

where $\mathbf{C}_{even}^{(j)}$ and $\mathbf{C}_{odd}^{(j)}$ are the two sub-vectors consisting of even and odd numbered samples, respectively, in the padded vector of $\mathbf{C}^{(j)}$.

It is seen from (3.6) that the operations in each of the two terms are identical, and also, they can be performed independently in parallel. Furthermore, in view of the structures of the decomposed transform matrices as given by (3.5), the filtering operation can be carried out by employing the conventional clocking mechanism used for implementing digital systems.

3.2 Choice of a Pipeline for the 1-D DWT Computation

In a pipeline structure for the DWT computation, multiple stages are used to carry out the computations of the various resolution levels of the transform. Thus, the computation corresponding to each resolution level needs to be mapped to a stage or stages of the pipeline. In order to maximize the hardware utilization of a pipeline, the hardware resource of a stage should be proportional to the amount of the computation assigned to

the stage. Since the amount of computations in successive resolution levels of the transform get reduced by a factor of two, two scenarios can be used for the distribution of the computations to the stages of a pipeline. In the first scenario, the resolution levels are assigned to the stages so as to equalize the computations carried out by each stage, that is, the hardware requirements of all the stages are kept the same. In the second scenario, the computations of the successive resolution levels are assigned to the successive stages of a pipeline, on a one-level-to-one-stage basis. Thus, in this case, the hardware requirement of the stages gets reduced by a factor of two as they perform the computations corresponding to higher-level decompositions.

Fig. 3.1 shows a stage-equalized pipeline structure, in which the computations of all the $K=\log_2 N$ levels are distributed equally among the M stages. The process of stage equalization can be accomplished by dividing equally the task of a given level of decomposition into smaller subtasks and assigning each such subtask to a single stage and/or by combining the tasks of more than one consecutive level of decomposition into a single task and assigning it to a single stage. Note that generally a division of the task would be required for low levels of decomposition and a combination of the tasks for high levels of decomposition.

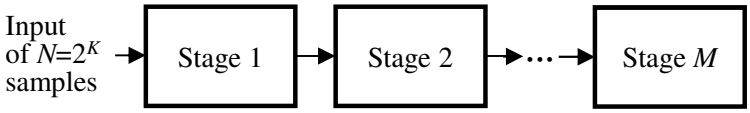


Figure 3.1: Stage-equalized pipeline structure.

In a one-to-one mapped structure, the computations of K resolution levels are distributed exactly among K stages, one level to one stage. In practical applications, a structure with less than K stages is used for the computation of a K-level DWT, as shown

in Fig. 3.2. In this structure, the computations of the first $I-1$ levels are carried out by the stages $i=1, 2, \cdots, I-1$, respectively, and those of the last $K-I+1$ levels are performed recursively by the Ith stage. The amount of hardware resources of a stage is one-half of that of its preceding one except for the Ith stage that has the same size as that of the preceding stage.

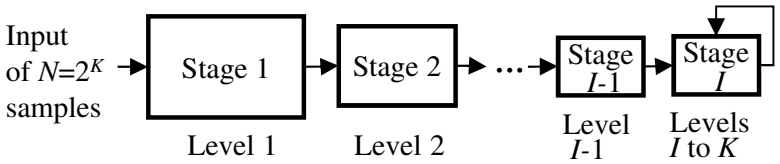


Figure 3.2: A one-to-one mapped pipeline structure with I ($I<K$) stages.

The structures of Fig. 3.1 and Fig. 3.2 can be used to perform the computations of multiple levels of decomposition. The computation of each level is performed as an L-tap FIR filtering operation by summing the L products of the input samples and the filter coefficients, as described by (3.2). Generally, one MAC cell is used to carry out one multiplication of an input sample by a coefficient followed by one accumulation operation. In order to perform an uninterrupted L-tap filtering operation with easy control, one can thus use a network of L basic units of such a MAC cell. Since all the resolution levels perform L-tap filtering operations, it would be desirable that each resolution level performs its filtering operation using this same type of MAC-cell network. However, in the context of one-to-one mapped pipeline structure of Fig. 3.2, in which the requirement is that the hardware resource should get reduced by a factor of two from one stage to the next, the use of the same MAC-cell network for all the stages would not be possible unless the pipeline has only two stages. In other words, the first stage performs the level-1 computation and the second stage performs the computations corresponding to all

the remaining levels recursively. In the context of a stage-equalized pipeline structure of Fig. 3.1, where the requirement is that all the stages should have the same hardware resource, the same MAC-cell network can be used easily for all the stages. However, in this case, the same amount of the computations cannot be assigned to all the stages that are based on the same MAC-cell network unless again there are only two stages in the pipeline.

In a situation of a pipeline of more than two stages, each based on a network of L MAC cells, one cannot achieve a resource-efficient architecture. Thus, for either pipeline structure, i.e., the one-to-one mapped or stage-equalized, a two-stage pipeline would be the best choice in terms of the hardware efficiency as well as from the standpoint of design and implementation simplicity. Note that the two-stage version of either pipeline structure is the same and it is shown in Fig. 3.3. An additional advantage of the two-stage pipeline is in the design flexibility of a MAC-cell network where the multiplication and accumulation operations can be furnished together by using logic gates. These logic gates could be arranged into more efficient arrays yielding a shorter propagation delay for the MAC-cell network. Based on the above discussion, it seems logical to use the two-stage pipeline structure of Fig. 3.3 for the design and implementation of an architecture for the 1-D DWT computation. The next section is concerned specifically with a detailed design of the architecture.

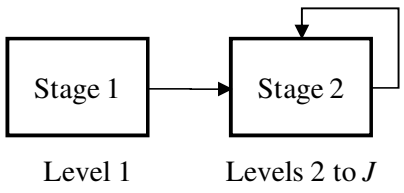


Figure 3.3: Pipeline structure with two stages.

3.3 Design of the Architecture

In the previous section, we advocated a two-stage pipeline structure for the computation of the 1-D DWT. The structure, whose development is constrained by the nature of the DWT computation, is capable of optimizing the use of hardware resources. In this two-stage structure, stage 2 performs by operating on the data produced by stage 1 as well as on those produced by itself, and therefore, the operations of the two stages need to be synchronized in a best possible manner [133]. In this section, we present the design of the proposed two-stage pipeline architecture focusing on data synchronization, the details of the various components comprising the stages, and inter and intra stages data flow.

3.3.1 Synchronization of Stages

In order to develop a suitable synchronization scheme, consider the timing diagram for the relative operations of the two stages shown in Fig. 3.4, where t_1 and t_2 are the times taken individually by stage 1 and stage 2, respectively, to carry out their operations, and t_a and t_c are the time spans during which stage 1 or stage 2 alone is operational, and t_b is the overlapped time span for the two stages. Our objective is to minimize $t_a+t_b+t_c$. Since the operation of stage 1 is independent of that of stage 2, it can continue its operation continuously until the computation of all the samples of resolution level 1 are computed. In Fig. 3.4, the slots shown for stage 1 correspond to $N/2$ samples of resolution level 1 that it has to compute. The presence of continuous slots indicates that stage 1 can continue its operation uninterruptedly without having any idle slot. Thus, the minimal possible value for t_1 is equal to $N \cdot T_c/2$, where T_c is the time required to compute one

output sample. If $J=\log_2 N$ and we assume that the DWT operation has to be carried out for all the J levels, then the number of samples that stage 2 has to compute is $N/2-1$. Thus, the lowest bound for t_2 is $(N/2-1)T_c$. Now, by choosing a value of t_c equal to its lowest bound, if one can show that $t_2=t_1-T_c$ (i.e. stage 2 does not have any idle slot during t_2), then indeed not only $t_a+t_b+t_c$ will be minimized but one also achieves its lowest bound. Now, we will show that for the proposed architecture this is so possible.

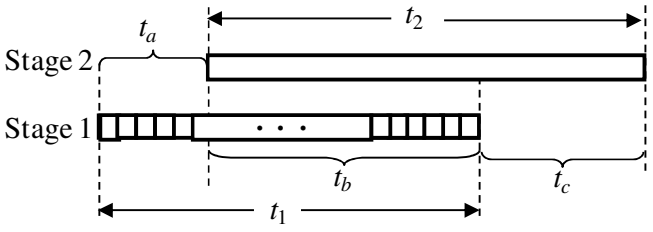


Figure 3.4: Timing diagram for the operations of two stages.

Let us first determine the lowest bound on t_c. Since the last sample of level 1 as produced by stage 1 becomes available only at the end of t_b, a sample at level $j\geq 2$ that depends on this last sample directly or indirectly could not possibly be computed during the time span t_b, and therefore, has to be computed during t_c. Assume that (i) during t_c we compute n_c samples of levels 2 and higher, which could not possibly be computed during t_b, and (ii) other output samples necessary for computing those n_c samples have already been computed during t_b. The lowest bound on t_c is n_cT_c. Therefore, in order to compute this bound, we need to determine the value of n_c. The last sample of level 1, which is computed at the end of t_b, is $C_{N/2}^{(1)}$. There are $k=\lceil L/2 \rceil$ output samples at level 2 that depend on this sample and they are given as $C_i^{(2)}$, $i=\lfloor (2^{J-1}-L+2)/2 \rfloor,\ldots,2^{J-2}$, where $\lceil x \rceil$ and $\lfloor x \rfloor$ represent the smallest integer larger than or equal to x and the largest integer less than or

equal to x, respectively. Next, at level 3, there are $\lceil (k+L-2)/2 \rceil$ output samples that indirectly depend on $C_{N/2}^{(1)}$ and they are given as $C_i^{(3)}$, $i=\lfloor (2^{J-2}-k-L+4)/2 \rfloor, \ldots, 2^{J-3}$. Similarly, we can determine the numbers and samples that depend indirectly on $C_{N/2}^{(1)}$ for other levels. Table 3.1 givens the listing of the numbers and samples of levels from j=2 to J that depend on $C_{N/2}^{(1)}$. After adding the expression in the third column of this table and some manipulation, it can be shown that the value of n_c can be obtained as

$$n_c = \left\lceil \frac{L}{2} \right\rceil + \sum_{j=3}^{J-\lceil \log_2 L \rceil} \left\lceil \left(\left\lceil \frac{L}{2} \right\rceil + \left(2^{j-2}-1\right)L - 3 \cdot 2^{j-3} + 1 \right) \Big/ 2^{j-2} \right\rceil + \left(2^{\lceil \log_2 L \rceil} - 1\right) \tag{3.7}$$

In Fig. 3.4, t_a is chosen to be $(n_c+1)T_c$. Next, we explore the possibility of developing a synchronization scheme for computing all the output samples in the context of Fig. 3.4 with the objective that stage 2 does not create any idle slots. In developing such a scheme, one has to take into consideration, the requirement of the underlying filtering operation of

Table 3.1: Indices and numbers of samples computed in time t_c

Level	Indices of samples $C_i^{(j)}$	Number of samples
2	$i = \left\{ \lfloor 2^{J-2} - L/2 + 1 \rfloor, \cdots, 2^{J-2} \right\}$	$k = \lceil L/2 \rceil$
3	$i = \left\{ \lfloor 2^{J-3} - (k+L-4)/2 \rfloor, \cdots, 2^{J-3} \right\}$	$\lceil (k+L-2)/2 \rceil$
⋮	⋮	⋮
j	$i = \left\{ \left\lfloor 2^{J-j} + \frac{L-k-1}{2^{j-2}} + \frac{5-2L}{2} \right\rfloor, \cdots, 2^{J-j} \right\}$	$\left\lceil \frac{k-L+1}{2^{j-2}} + \frac{2L-3}{2} \right\rceil$
⋮	⋮	⋮
J	$i = 2^{J-J}$	1

the wavelet computation. This filtering operation imposes the constraint that the first output sample at level j cannot be computed until L samples at level $j-1$ have already been computed and each of the subsequent samples at level j cannot be computed unless two new samples at level $j-1$ have already been computed. Note that this requirement of the filtering operation imposes a constraint on the operation of stage 2 only, since stage 1 operates sequentially and unilaterally to compute the level-1 output samples only. Under this constraint, we now give three steps of the synchronization that govern the computation of the output samples at various resolution levels by stage 1 and 2.

Step 1: Stage 1 operates continuously to compute the level-1 output samples sequentially.

Step 2: Stage 2 starts the computation of level-2 samples beginning at the time slot (n_c+2).

Step 3: (*a*) *When stage 2 is computing an output sample at the lowest incomplete level j≥2.* After completing the computation of the present sample at this level stage 2 moves on to the computation of a sample at the lowest higher level, if the data required for the computation of this sample have become available; otherwise stage 2 continues with the computation of the next sample at the present level j.

(*b*) *When stage 2 is computing an output sample at a level other than the lowest incomplete level.* After completing the computation of the present sample, stage 2 moves its operation to the lowest incomplete level.

The rationale behind *Step* 3(*a*) is that moving the operation of stage 2 to a higher level allows more data from level 1 as produced by stage 1 to become available, since the availability of the output samples of level 1 is crucial for the computation of the samples at higher levels. On the other hand, the rationale behind *Step* 3(*b*) is that there are always more samples to be computed at lower levels than that at higher levels, and therefore, more time needs to be spent in computing lower level samples.

The nature of the filtering operation coupled with the decimation by a factor of 2 requires that, in order for stage 2 to compute a level-2 sample at slot m, stage 2 needs L level-1 samples computed by stage 1 at slots i+1, i+2,…, i+L (i<m−L), of which the samples produced at the last two slots must not have been previously used for the computation of level-2 samples. If stage 2 can meet this requirement during the entire time span t_b, then it can continue its operation uninterruptedly without creating an idle slot. We will now show that, based on the steps presented above, stage 2 would indeed be able to meet this requirement. For this purpose, consider an algorithm, Algorithm 1, which synchronizes the operation of stage 2 during the time span t_b. In this algorithm, we have made use of two counters, namely p and q. The counters p and q represent the total number of samples having been computed at level 2 and that at the levels higher than 2, respectively, at a particular instant of stage-2 operation. Note that at the time that stage 2 starts its operation, stage 1 has already produced n_c+1 level-1 samples. Since a length-L filtering operation would require L input samples and (n_c+1)>L, stage 2 not only can start the computation of level-2 samples, but it can continue the computation of the succeeding level-2 samples at least for some time. Since the computation of each level-2 sample makes use of two new level-1 samples during the time in which only one level-1 sample

is produced by stage 1, the number of available level-1 samples is reduced by one after the computation of each level-2 sample. However, since stage 2, following *Step* 3 of the synchronization, is allowed to compute the samples at levels higher than 2 without making use of the samples from level 1, the reservoir of level-1 samples is increased by one after the computation of one such a higher-level sample. Therefore, at a particular time, there are $n_b=n_c+1-(p-q)$ level-1 samples available to be used by stage 2 for the computation of the succeeding level-2 samples. Since p increases faster than q, $p-q$ reaches its maximum value at the time slot just before the end of the time span t_b. At this time slot,

$$p = \lfloor 2^{J-2} - L/2 \rfloor \tag{3.8a}$$

$$q = \sum_{i=3}^{J} \left\lfloor \frac{2^{J-2} - \lceil L/2 \rceil - (2^{i-2}-1)L + 3\cdot 2^{i-3} - 1}{2^{i-2}} \right\rfloor \tag{3.8b}$$

Thus, using (3.7), (3.8a) and (3.8b), the lowest bound of n_b during the time span t_b is calculated as

$$n_b \geq n_c + 1 - \left(\left\lfloor 2^{J-2} - \frac{L}{2} \right\rfloor - \left(2^{J-2} - 1 - n_c + \left\lceil \frac{L}{2} \right\rceil \right) \right) \tag{3.9}$$

Since in practice the filter length L is such that $L<2^{J-1}-1$, the above inequality can be written as

$$n_b \geq \left\lceil \frac{L}{2} \right\rceil - \left\lfloor -\frac{L}{2} \right\rfloor = \begin{cases} L & \text{if } L \text{ is even} \\ L+1 & \text{if } L \text{ is odd} \end{cases} \tag{3.10}$$

Thus, the lowest bound on n_b is greater than or equal to L. Therefore, during the entire course of the time span t_b, there will always exist sufficient number of samples available to stage 2 for it to continue its level-2 computation in the frame work of Algorithm 1. In

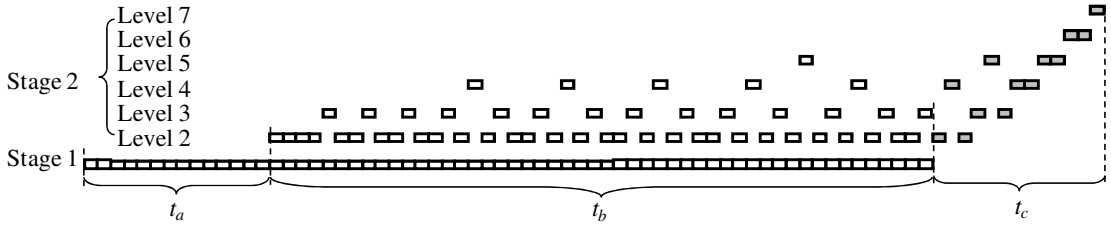


Figure 3.5: Synchronization scheme for a 128-point (J=7) DWT computation using length-4 (L=4) FIR filter.

other words, during the time span t_b, stage 2 would never have to cease its operation for the lack of availability of at least 2 new level-1 samples, that is, the block in Algorithm 1 that introduces a unit delay T_c will never be used during the execution of the algorithm.

Algorithm 1: Synchronizing the operation of stage 2 during t_b
Initialize $p \leftarrow 0$, $q \leftarrow 0$
While $p + q \leq 2^{J-1} - n_c$
    If (at least 2 new samples available from level 1) **then**
        Compute a new sample at level 2
        $p \leftarrow p + 1$
        If (enough data available from the lowest level $k \geq 2$) **then**
        Compute a new sample at level $k + 1$
        $q \leftarrow q + 1$
        End if
    Else
    Unit delay T_c
    End if
End while
End algorithm

We now consider an example to illustrate the synchronization scheme that has been presented above. For this purpose, we consider a 128-point (J=7) DWT computation using 4-tap (L=4) FIR filters. The synchronized operation of the two stages is shown in Fig. 3.5, in which each rectangle represents a time slot during which a lowpass output sample is produced. Stage 1 starts the computation of the first level-1 output sample at slot 1 and continuous its operation until slot 64 when the computation of the 64th level-1 output sample is completed. Equation (7) can be used to obtain the value of n_c as 13. Thus, at the slot number (n_c+2)=15, stage 2 starts the computation of the first level-2 output sample. At this point, the reservoir of level-1 available samples contains (n_c+1)=14 samples. Note that the number of samples in this reservoir decreases by one sample as one new level-2 sample is computed and it increases by one as one sample at a level higher than 2 is computed. However, the general trend is a decline in the number of available level-1 samples from 14 samples at slot 15 to 4 samples at slot 65 when the computations of all level-1 samples are completed. At slot 66, an output sample at level 4 is computed, since the required samples from level-3 have become available for its computation. After this computation, stage 2 returns its operation to the computation of the last level-2 output sample. Note that for the computation of this last level-2 sample, two padded samples would be required, since at this time no level-1 output sample is unused. Beyond this point, all the remaining samples from level 3 to level 7 are computed using *Step* 3 of the synchronization.

3.3.2 Design of Stages

Since in the stage-equalized architectures, the two stages together perform the DWT computation with amount and the type of computations of the individual stages being the same, each of the two stages can use identical processing units. However, the control units to be employed by the stages have to be different, since, as seen from Algorithm 1 of the previous subsection, the operation of stage 1 is autonomous, whereas stage 2 must always synchronize its operation with that of stage 1. Based on this algorithm, the design of the control unit used by stage 2 would have to be a bit more involved than that of the control unit used by stage 1. Obviously, in order to synchronize the operation of stage 2 with that of stage 1, a buffer has to be used to store the lowpass output samples from the two stages. Fig. 3.6 gives a block diagram incorporating all these requirements for the design of the proposed architecture. The two processing units are referred as PU_1 in stage 1 and PU_2 in stage 2. Note that in this architecture, the highpass samples from PU_1 and PU_2 are outputted directly.

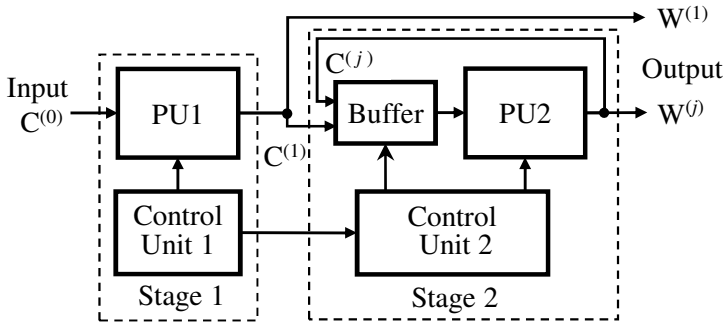


Figure 3.6: Block diagram of the two-stage architecture.

In each stage, the processing unit by employing L multiplication-and-accumulation (MAC) cells network performs an L-tap filtering operation and at each clock cycle

generates a total of L product terms and their sum. Since, normally, the interval between the two consecutive input samples must not be smaller than the delay of a MAC cell, the maximal allowable data rate of the input to the processing unit would be determined by this delay. However, if the L-MAC-cell network is organized into m sub-networks operating in parallel, the input samples can be applied to these sub-networks in an interleaved manner. The interval of the two consecutive input samples can thus be shortened by a factor m. To this end, considering the problem at hand in which a two-subband filtering operation is performed and for each consecutive resolution level the input data is decimated by a factor of 2, the L MAC cells can be conveniently organized into a pair of even and odd filter blocks. These even and odd filter blocks, which receive the even and odd numbered input samples, respectively, employ $L/2$-MAC-cell networks, and each produces only $L/2$ product terms and their sums. The partial sums from the two networks are required to be added in an accumulation block by using a carry propagation adder (CPA), as shown in Fig. 3.7. Since the delay of the accumulate block is comparable to that of the $L/2$-MAC-cell network, it is useful to pipeline them for parallel computation. Since the high-pass operation differs from that of the low-pass operation only in reversing the sign of the even-numbered coefficients, the proposed organization of the processing unit would allow the filter block to use the same filter coefficients simply by introducing a sign inversion block into the even filter block.

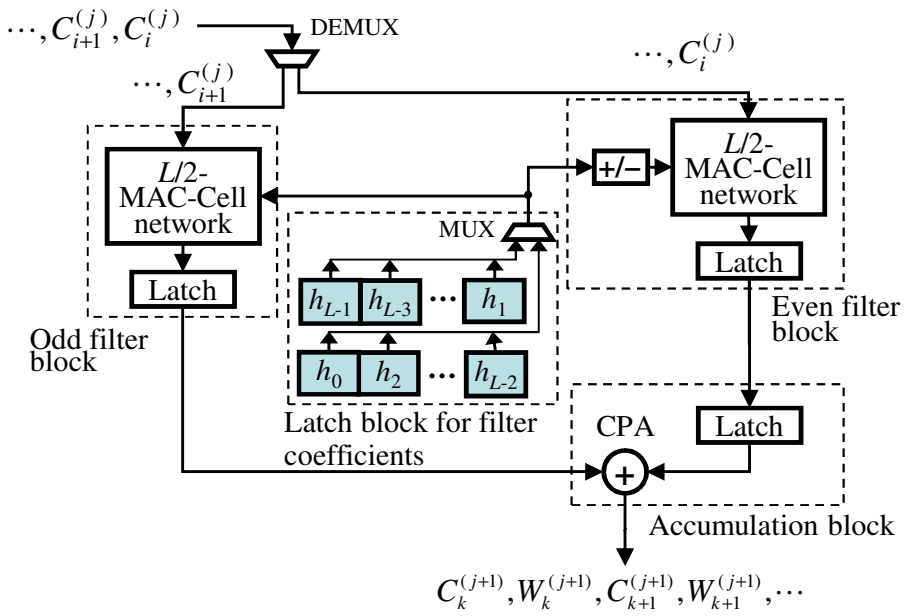


Figure 3.7: Block diagram of the processing unit for L-tap filtering computation assuming L to be an even number.

As discussed earlier and seen from Fig. 3.6, all the output data must be synchronized in accordance with Algorithm 1. This synchronization process is facilitated by introducing in stage 2 a buffer, which stores output data from the two stages and provides input data to stage 2. According to *Step* 2 of the synchronization scheme, during the time span t_a, the number of samples that need to be stored for the operation of stage 2 increases until n_c+1. However, this number will not exceed n_c+1 during the time spans t_b and t_c, since the number of samples newly produced by stage 1 and 2 is equal to or less than that consumed by stage 2. Thus, the minimum capacity of the buffer for the operation of stage 2 is n_c+1 registers. Since the number of output samples at a level that would be needed to compute an output sample at the next higher level will not exceed the filter length L, the buffer, therefore, is divided into $k=\lfloor (n_c+1)/L \rfloor$ channels, as shown in Fig. 3.8. Each channel consists of L shift registers except channel k that only has (n_c+1

mod L) registers, where (a mod b) is the remainder on division of a by b. Channel 1 is used for storing only the level-1 samples produced by PU_1, whereas channel j=2,...,k for the level-j samples during t_b and t_c, and would also be used for storing the level-1 samples during t_a. Note that channel 2 is also chosen to store the samples of the remaining levels $j \geq k$ since the time slot that all the level-2 samples have been consumed.

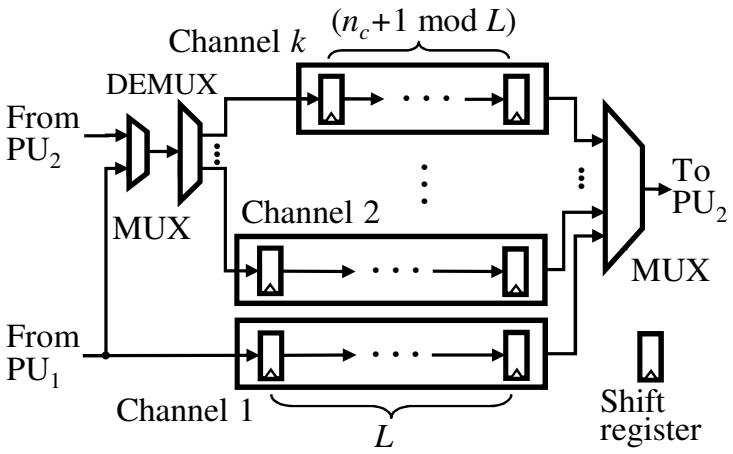


Figure 3.8: Structure of the buffer.

3.3.3 Design of L/2-MAC-cell Network

In the processing unit shown in Fig. 3.7, each physical link from a given input bit to an output bit of an L/2-MAC-cell network gives rise to a channel or data path having a delay that depends on the number and the types of operations being carried out along that path [133]. Thus, it is crucial to aim at achieving the shortest critical data path when designing an L/2-MAC-cell network for our architecture. In order to have a better appreciation of the operations of an L/2-MAC-cell network, let us consider an example of the filtering operation of one such network with L/2=2. Let us assume that the input samples and the filter coefficients have the wordlengths of 6 and 3, respectively. Each

MAC-cell network has 6 partial products, with a total of 36 bits, which can be produced in parallel, as shown in Fig. 3.9(a). Our objective is to design a MAC-cell network, in which the bits of the partial products are accumulated in such a way as to optimize the delays of the data paths from the individual bits of the partial products to the output bits of the MAC-cell network.

Even though all the bits of the partial products as given by the array shown in Fig. 3.9(a) are available simultaneously, they cannot be used in parallel to produce simultaneously all the bits of an output sample. The reason for this is that the processes of accumulation of the bits in each column of the array of the partial products have to be carried out bit-wise and at the same time one has to take care of the propagations of the carry bits. In other words, the accumulation of the partial products has to be carried out in a certain sequence. Thus, the task of accumulation can be divided into a sequence of layers such that the operations of the first layer depend only on the partial products bits and those of the succeeding layers depend on the partial product bits not yet used as well as on the bits of the results of the preceding layers. In order to meet our goal of minimizing the critical path from a partial product bit to a bit of the output sample, we can organize the layers of the MAC-cell network that would carry out the accumulation of the partial products based on the following guiding principle. Minimize the number of layers while minimizing the delay of each layer. The number of layers can be minimized by assigning to each layer the maximum number of such tasks that can be performed independent of each other in parallel. The accumulation task in each layer can be performed by using full-adder (3:2) and double-adder (2×2:3) modules, as shown in Fig. 3.9(b). The two types of module are chosen, since (i) their delays are about the same so

that the delay of any layer can be made to be equal to this delay irrespective of whether the layer uses one type or two types of modules, and (ii) the two modules can be used together in such a way so that they produces of a smaller number of the propagating carry bits, and therefore, their combined use helps in reducing the number of layers.

With the choice of the combination of the full-adders and double-adders, the first layer can be formed by using as many modules as necessary with the maximum number of partial product bits being utilized as 3-bit or 4-bit inputs to the respective modules. Scanning the partial product array from right to left, a maximum number of bits of this array are first used as inputs to as many full-adder modules as necessary, since in comparison to a double-adder this module is more efficient in consuming the bits of the input array. In this process, whenever in a column (i) only two bits of the partial product array are left unused, these two bits along with a pair of bits from the neighbouring left column of the array are used as inputs to a double-adder modules, and (ii) only one bit of the partial product array is left unused, then this bit is used in the next layer for accumulation. Note that the case of using a double-adder also helps in propagating two carry bits, one internal and the other external to the adder, to the left within the same time delay as that of the full-adder. The next layer can then be formed again by using as many modules as necessary with inputs from the partial product bits, still unused, and the sum and carry output bits from the previous layers being utilized in a carry-save manner. This process can be continued until after the last layer when all the bits of an output sample are produced.

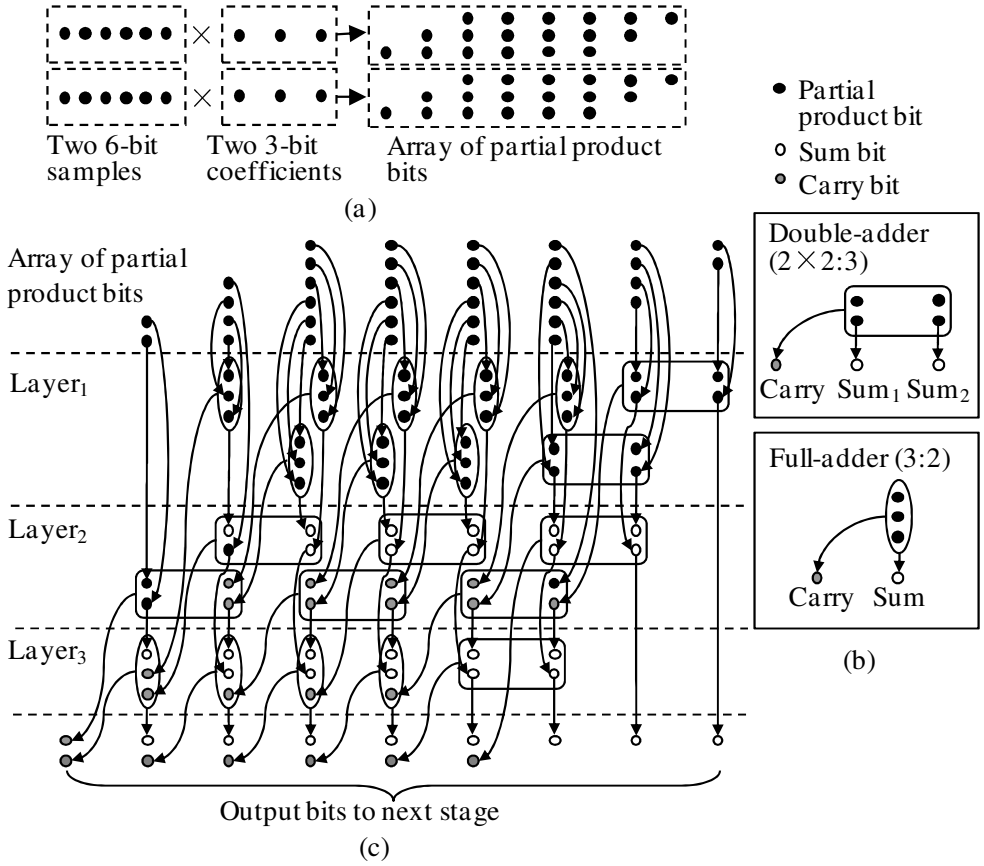


Figure 3.9: A two-dimensional array of bit-wise additions. (a) Formation of an array of partial products. (b) Two types of bit-wise adders. (c) A layered organization of bit-wise addition using the two modules in (b).

Based on the principles and the procedure enunciated above, we can now give formally an algorithm, Algorithm 2, which carries out the organization of a MAC-cell network, given $L/2$ input samples and $L/2$ filter coefficients. Fig. 3.9(c) gives an illustration of the organization of the adder modules into three layers of a MAC-cell network for the example considered earlier. It is seen from this figure that the delay of the critical path is equal to that of three full-adders for this particular example.

Algorithm 2: Organizing the bit-wise modules of the MAC-cell network

Initialize an $N_I(k)\times M_I$ array $\mathbf{A_I}$ of partial product bits from the $L/2$ X-bit samples and $L/2$ Y-bit filter coefficients, where $M_I=X+Y-1$ and

$$N_I(k)=\begin{cases} kL/2 & 1\le k\le \min(X,Y)-1 \\ \min(X,Y)\cdot L/2 & \min(X,Y)\le k\le \max(X,Y)-1 \\ (X+Y-k)\cdot L/2 & \max(X,Y)\le k\le M_I \end{cases}$$

While $N_I(k)\ge 3$ for any $1\le k\le M_I$

Initialize the elements of an $N_O(k)\times(M_I+1)$ array $\mathbf{A_O}$ by $N_O(k)\leftarrow$ *zeros* for $k=1,\ldots,M_I+1$

For every column $i=M_I,\ldots,2,1$

While $N_I(i)\ge 3$

Assign 3 bits, $A_I[N_I(i)--,i]$, $A_I[N_I(i)--,i]$, $A_I[N_I(i)--,i]$, as inputs to a full-adder

Append one sum bit to $A_O[++N_O(i),i]$, and one carry bit to $A_O[++N_O(i-1),i-1]$ in $\mathbf{A_O}$

End while

If $N_I(i)=2$ and $N_I(i-1)\ge 2$ **then**

Assign 2×2 bits, $A_I[N_I(i-1)--,i-1]$, $A_I[N_I(i-1)--,i-1]$, $A_I[N_I(i)--,i]$, $A_I[N_I(i)--,i]$, as inputs to a double-adder

Append two sum bits to $A_O[++N_O(i),i]$, $A_O[++N_O(i-1),i-1]$, and one carry bit to $A_O[++N_O(i-2),i-2]$ in $\mathbf{A_O}$

Else

Carry forward unused bits $A_I[N_I(i)--,i]$ to $A_O[++N_O(i),i]$ in $\mathbf{A_O}$

End if

End for

$\mathbf{A_I}\leftarrow\mathbf{A_O}$

End while

End algorithm

Using Algorithm 2, a generalized structure for the MAC-cell network, as shown in Fig. 3.10, can be generated with $L/2$ X-bit samples and $L/2$ Y-bit filter coefficients as inputs to the network. Layer_0 produces a total of $X \cdot Y \cdot L/2$ partial product bits. The accumulations of these partial product bits are carried out successively by a set of layers of adder modules. A variable size array is used as input to each layer. This array initially contains only the partial product bits, and for successive layers, it contains the sum and carry bits from the previous layers and the partial product bits still unused. An input to a layer that consists of a partial product bit or a sum bit is shown in the figure by an arrow going down vertically into the layer, whereas an input that consists of a carry bit is shown by an arrow going down leftward. The MAC-cell network has a total of $Z = \lceil \log_{3/2}[\min(X,Y) \cdot L/4] \rceil$ layers, which is the minimum number of layers with the choice of using the maximum number of full-adders followed by, if necessary, the double-adders in each layer. The number of adder modules used for each layer progressively decreases from Layer_0 to Layer_Z. The output bits of the MAC-cell network are then used by the accumulation block of the processing unit to produce the final sum. In above design of the MAC-cell network, optimization of its critical path is carried out by incorporating and arranging the multiply and accumulate operations into multiple layers. This leads to a network that has a critical path with a smaller delay than the delay of the MAC cell used in DSP processors, in which the delay of the critical path is simply the sum of the delays associated with a multiplier and an accumulator. The critical path of the MAC-cell network could be shortened further by encoding the input data to the MAC-cell network using booth encoders. Thus, the delay of the MAC-cell network is reduced by making a smaller number of carry bits to propagate through the MAC-cell network. However, such

an improvement can be achieved with an expense of additional hardware resources to be used for encoders.

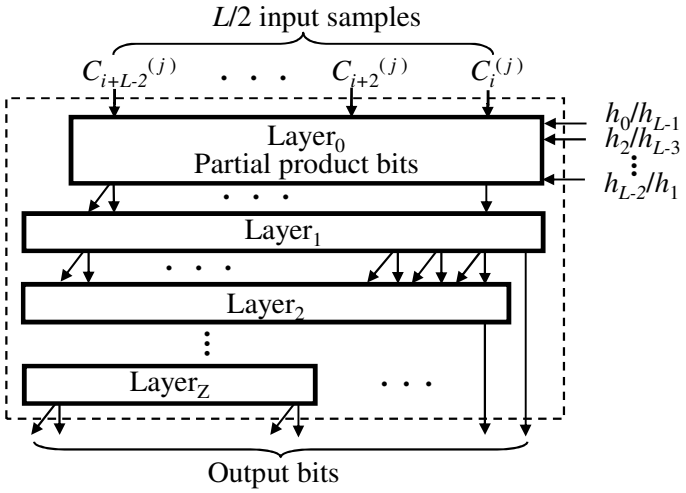


Figure 3.10: Structure of the *L*/2-MAC-cell network.

3.4 Performance Evaluation and FPGA Implementation

In order to evaluate the performance of the architecture resulting from the proposed scheme, we need to make use of certain metrics that characterize the architecture in terms of the hardware resources used and the computation time. The hardware resources used for the filtering operation are measured by the number of multipliers (N_{MUL}) and the number of adders (N_{ADD}), and that used for the memory space and pipeline latches is measured by the number of registers (N_{REG}). The computation time, in general, is technology dependent. However, a metric, that is independent of the technology used but can be utilized to determine the computation time T, is the number of clock cycles (N_{CLK}) consumed from the instant the first sample is inputted to the last sample outputted assuming a given clock cycle period, say unity, as the latency of a MAC cell.

Table 3.2: Comparison of various architectures

Architecture	N_{MUL}	N_{ADD}	N_{REG}	N_{CLK}
Parallel [53]	$2L$	$2L-2$	$JL+4L$	$N+JL$
Systolic [43]	L	$L-1$	$2JL+L+2$	$2N+2JL$
Pipelined [61]	$\sum_{k=1}^{J}\left\lceil \frac{L}{2^{k-2}} \right\rceil$	$\sum_{k=1}^{J}\left\lceil \frac{L}{2^{k-2}} \right\rceil$	$2JL+J+\sum_{k=1}^{J}\left\lceil \frac{L}{2^{k-2}} \right\rceil$	$N/2+JL/2$
DRU [62]	$\sum_{k=1}^{J}\left\lceil \frac{L}{2^{k-1}} \right\rceil$	$\sum_{k=1}^{J}\left\lceil \frac{L}{2^{k-1}} \right\rceil-1$	$JL+2J+\sum_{k=1}^{J}\left\lceil L/2^{k-1} \right\rceil$	$N+2J$
IP core [55]	$J\cdot\lceil L/4\rceil$	$J\cdot\lceil L/2\rceil$	$2JL+2J$	$N+JL$
Proposed	$2L$	$2L-2$	$4L+n_c+1$	$N+J$

For a J-level DWT computation of an N-sample sequence using L-tap filters, the expressions for the metrics mentioned above for various architectures are summarized in Table 3.2. Assuming that the number of samples N is much larger than $J{\cdot}L$, it is seen from the table that compared to the architecture of [61], all the other architectures, including the proposed one, require approximately twice the number of clock cycles, except the architecture of [43], which requires four times as many clock cycles. This performance of [61] is achieved by utilizing the hardware resources of adders and multipliers that is four times that required by the architecture of [43] and twice that required by any of the other architectures. However, if the value of $J{\cdot}L$ cannot be neglected in comparison to that of N, the values of N, J and L should be taken into consideration while comparing the architectures in terms of N_{CLK}. In this regard, only for the proposed architecture and the architecture of [62], N_{CLK} is independent of the filter length with the proposed architecture giving the lowest value of N_{CLK} for a given N. The proposed architecture requires the number of registers that is at least 20% less than that required by any of the

other architectures when the filter length L is large. It should be noted that approximately 20% of the hardware resource comprises registers.

Since the area of the circuit for the DWT computation depends on the filter length L and the total number of samples N, it would be useful to have a measure of the area of the circuit as functions of L and N. Only the proposed architecture and those of [53] and [62] are used for this study, since the numbers of multipliers and the numbers of adders for these architectures are the same. Thus, any difference in the areas of the three architectures could be accounted for due mainly to the difference in the number of the registers used by each of the architectures. As seen from Table 3.2, the number of registers for the architecture of [53] is $(J+4)L$ and that for the architecture of [62] is approximately $JL+2J+2L=(J+2)L+2J$. However, the number of registers for the proposed architecture not only depends directly on the filter length L but also indirectly on L and N through the parameter n_c. These dependencies are intuitively obvious from the fact that as the filter length or the number of samples increases, the starting point of stage 2 gets more delayed. In other words, n_c is increased. However, it is seen from this figure that the dependence of n_c on N is relatively much more non-linear than its dependence on L. The results of Fig. 3.11 can be used to obtain a measure of the area of the proposed architecture as functions of L and N. We estimate the areas of the proposed architecture along with that of the other two architectures under the assumption that the ratio of areas of one multiplier, one adder and one register is 12:3:1. The plots of the estimates of the areas as functions of L and N are shown in Fig. 3.12. It is obvious from this figure that area of the proposed architecture is, in general, lower than those of the other two architectures. The lower area of the proposed architecture can be attributed due mainly to

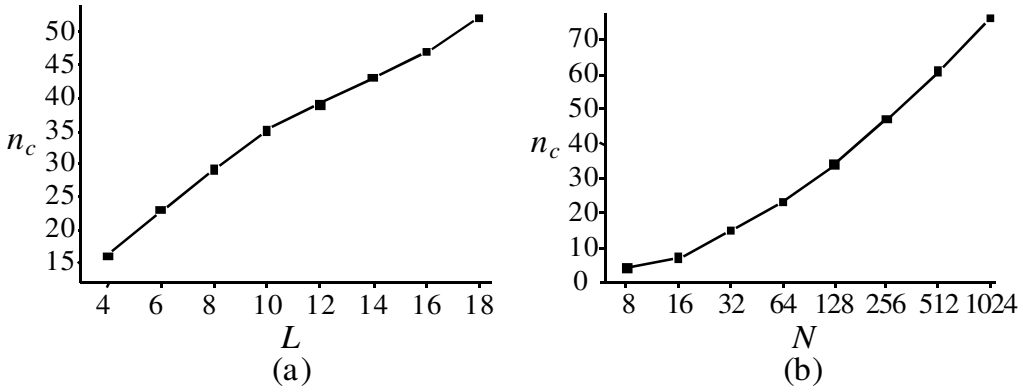


Figure 3.11: Estimated values of n_c. (a) n_c versus filter length L (N=2^8), and (b) n_c versus signal length N (L=16).

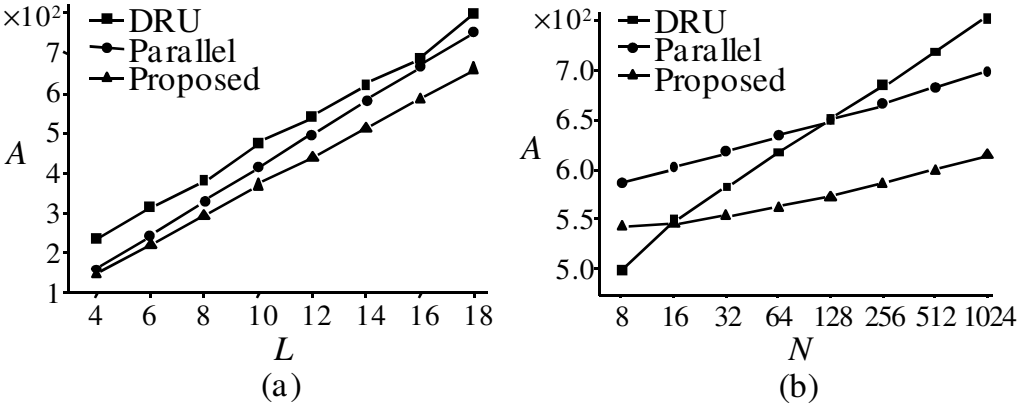


Figure 3.12: Estimated areas of the three architectures. (a) Area A versus filter length L (N=2^8), and (b) area A versus signal length N (L=16).

the presence of the parameter n_c in its expression for the N_{REG}. Recall that n_c is a parameter that we minimized in the design of the proposed architecture in order to maximize the parallelism between the two stages, and a lower value of n_c, in turn, results in smaller number of registers required to store the results of the operations of stage 1 before the operation of stage 2 starts.

Considering the clock cycle period T_c as the delay of the MAC cell used by an architecture, the computation time can be obtained as $T=N_{CLK}T_c$. Note that the reciprocal of T_c is simply the throughput of the architecture assuming that one sample is inputted during each clock cycle. Using T, one can determine the area-time complexity, AT, where the area, A, mainly comprises the areas of the multipliers, adders and registers. In order to evaluate the performance of the architectures in terms of T_c and AT, we consider an example of designing a circuit for the DWT computation where the sample size N=128 and the number of the resolution levels J=7. We use Daubechies 6-tap filter (L=6) as analysis filters and the sample and filter coefficient wordlengths are taken as 8 bits. The carry propagation adder of the processing unit utilizes the structure of a combination of carry-skip and carry-select adders [135]. The registers are designed using D-type flip-flops (DFF). All the modules, such as partial products generator, DFF, full-adder, double-adder, multiplexer and demultiplexer, used in the proposed architecture are designed by using 0.35-micron CMOS technology and simulated by using HSpice to obtain the delays. Note that these same modules are also used to evaluate the performance of all the other architectures. Table 3.3 shows the values of the clock cycle period and the area-time complexity for the various architectures. It is seen from this table that the proposed architecture has significantly smaller value of the clock cycle period compared to that of all the other architectures. The proposed architecture has the highest throughput of 138 MBPS (megabytes per second) and the lowest complexity in terms of area-time and area-$(\text{time})^2$, among all the architectures considered.

In order to estimate the power consumption of the proposed architecture, an example of the proposed architecture is constructed for a 7-level DWT computation of 8-bit

Table 3.3: Evaluation of various architectures

Architecture	T_c (ns)	$A \cdot T$	$A \cdot T^2$
Parallel [53]	17.8	243	4325
Systolic [43]	11.8	141	1664
Pipelined [61]	11.8	183	2159
DRU [62]	10.2	117	1193
IP core [55]	11.8	159	1876
Proposed	7.2	62	446

samples using 6-tap filters and simulated at a clock frequency of 138 MHz using Synopsys Power Compiler. The resulting power consumption values are 154.2 mW and 67.6 mW using 0.35-micron (V_{DD} = 3.3 V) and 0.18-micron (V_{DD} = 1.8 V) technologies, respectively.

In order to have a fair comparison of the power consumption performance of different architectures, the circuit complexities and the technologies used for the circuit design of the architectures under consideration must be the same. In this regard, estimates of the power consumption for the architectures listed in Table 3.3 are either unavailable or, if available, the underlying architectures have been designed with substantial differences in the circuit complexities and process technologies. Despite this difficulty in carrying out a fair comparison of power consumption of architectures, we compare the estimated power consumption of the proposed architecture with that given in [136]. The architecture of [136] is also a pipeline architecture that uses the same filter core as that used in [55] of Table 3.3. In [136], an example of the architecture using 9/3 filters and 9-bit samples has been constructed, and simulated for an operation at 100 MHz clock frequency using a 0.35-micron technology. The resulting power consumption figure is 325 mW. This value of power consumption is more than twice the value of 154.2 mW

obtained from the example of the proposed architecture in 0.35-micron technology, which is constructed by employing 6-tap filters operating on 8-bit samples at 138 MHz clock frequency.

In order to verify the estimated results for the example of the DWT computation considered above, an implementation of the circuit is carried out in FPGA. Verilog is used for the hardware description and Xilinx ISE 8.2i for the synthesis of the circuit on Virtex-II Pro XC2VP7-7 board. The FPGA chip consists of 36×36 arrays with 11,088 logic cells and it is capable of operating with a clock frequency of up to 400 MHz. The implementation is evaluated with respect to the clock period (throughput) measured as the delay of the critical path of the MAC-cell network, and the resource utilization (area) measured as the numbers of configuration logic block (CLB) slices, DFFs, look-up tables (LUTs) and input/output blocks (IOBs). The resources used by the implementation are listed in Table 3.4. The circuit is found to perform well with a clock period as short as 8.7 ns, a value that is reasonably close to the estimated value of 7.2 ns. The power consumption of the FPGA chip on which the designed circuit implemented is measured to be 105 mW (V_{DD}=1.5 V). Thus, the simulated value of 67.6 mW is reasonably realistic for power consumption for the circuit realizing the proposed architecture, considering the measured value of the power consumption also includes the power dissipated by the unused slices in FPGA.

Table 3.4: Resources used in FPGA devices

Resource	Number used	Total number available	Percentage used
CLB Slices	1532	4928	31%
Flip Flop Slices	858	9856	8%
4-input LUTs	2888	9856	29%
Bonded IOBs	38	248	15%

In order to further validate the proposed scheme, the FPGA implantation results of the proposed architecture are obtained and compared with those of existing architectures. The implementation results for the architectures given in [137]–[141] and for the proposed one are listed in Table 3.5. It is seen from this table that the proposed architecture provides the highest operational clock frequency with a hardware cost similar to that of the existing architectures.

Table 3.5: FPGA implementation results for various 1-D architectures

Architecture	Number of CLB/LE slices*	f_{max} (MHz)	Device
[137]	615	73 (L=4)	XC4036
[138]	369	26	XCV3000
[139]	422	94	XC300
[140]	837	14.8	Virtex V100
[141]	678	96.6	Stratix
Proposed	567	125	XC2VP30

* The slices excluding RAM

3.5 Summary

In this chapter, a scheme for the design of a pipeline architecture for real-time computation of the 1-D DWT has been presented. The objective has been to achieve a low computation time by maximizing the operational frequency ($1/T_c$) and minimizing the number of clock cycles (N_{CLK}) required for the DWT computation, which in turn, have been realized by developing a scheme for an enhanced inter-stage and intra-stage parallelisms for the pipeline architecture.

A study has been undertaken that suggests that, in view of the nature of the DWT computation, it is most efficient to map the overall task of the DWT computation to only two pipeline stages, one for performing the task of the level-1 DWT computation and the other for performing that of all the remaining resolution levels. In view of the fact that the amount and nature of the computation performed by the two stages are the same, their internal designs ought to be the same. There are two main ideas that have been employed for the internal design of each stage in order to enhance the intra-stage parallelism. The first idea is to decompose the filtering operation into two subtasks that operate independently on the even- and odd-numbered input samples, respectively. This idea stems from the fact that the DWT computation is a two-subband filtering operation, and for each consecutive resolution level, the input data are decimated by a factor of two. Each subtask of the filtering operation is performed by a MAC-cell network, which is essentially a two-dimensional array of bit-wise adders. The second idea employed for enhancing the intra-stage parallelism is to organize this array in a way so as to minimize the delay of the critical path from a partial product input bit to a bit of an output sample

through this array. In this chapter, this has been accomplished by minimizing the number of layers of the array while minimizing the delay of each layer.

In order to assess the effectiveness of the proposed scheme, a pipeline architecture has been designed using this scheme and simulated. The simulation results have shown that the architecture designed based on the proposed scheme would require the smallest number of clock cycles (N_{CLK}) to compute N output samples and a reduction of at least 30% in the period of the clock cycle T_c in comparison to those required by the architectures with a comparable hardware resource requirement. An FPGA implementation of the architecture designed has been obtained demonstrating the effectiveness of the proposed scheme for designing efficient and realizable architectures for the DWT computation.

Chapter 4

A Scheme for the Design of Pipeline Architectures for 2-D Discrete Wavelet Transform

As discussed in Chapter 2, the architectures for the computation of the 2-D DWT can be classified into separable and non-separable architectures. In a separable approach, the 2-D filtering operation of an architecture is divided into two 1-D filtering operations. A separable pipeline architecture for the computation of the 2-D DWT can easily be developed by using the scheme proposed in the previous chapter. However, the resulting architecture would have a large latency. Moreover, separable filters of this architecture would not be able to approximate well arbitrary frequency responses. On the other hand, a pipeline architecture using non-separable filters should provide more flexibility in providing low latency and in employing filters with arbitrary frequency responses.

In this chapter, a scheme for the design of fast pipeline architectures for the computation of the 2-D DWT based on the non-separable approach is developed [142], [143]. Even though the goal of fast computation is achieved by minimizing the number and period of clock cycles, the main ideas of the 1-D scheme of optimally distributing the task of the 2-D DWT computation and maximizing the inter-and intra-stage parallelisms

cannot be extended in a straightforward manner to case of non-separable 2-D architectures. The work of this chapter, while developing a scheme for the design of non-separable pipeline architectures for the computation of 2-D DWT, is specifically focused on optimally distributing the overall task of the 2-D DWT computation and on maximizing the inter- and intra-stage parallelisms of the pipeline.

The chapter is organized as follows. In Section 4.1, a mathematical formulation of the 2-D DWT computation necessary for the development of the proposed architecture is presented. In Section 4.2, a study is conducted to determine the number of stages of a pipeline necessary for optimally mapping the task of the DWT computation onto the stages of the pipeline. Based on this study, in Section 4.3, a three-stage pipeline architecture is developed with an efficient structure of the 2-D input data and an optimal organization of the processing units in each of the stages. In Section 4.4, the performance of the proposed architecture is assessed and compared with that of other existing architectures and validated by an FPGA implementation. Section 4.5 summarizes the work of this chapter and highlights the salient features of the proposed scheme.

4.1 Formulations for the Computation of the 2-D DWT

The 2-D DWT is an operation through which a 2-D signal is successively decomposed in a spatial multi-resolution domain by lowpass and highpass FIR filters along each of the two dimensions. The four FIR filters, denoted as highpass-highpass (HH), highpass-lowpass (HL), lowpass-highpass (LH) and lowpass-lowpass (LL) filters, produce, respectively, the HH, HL, LH and LL subband data of the decomposed signal at a given resolution level. The samples of the four subbands of the decomposed signal at

each level are decimated by a factor of two in each of the two dimensions. For the operation at the first level of decomposition, the given 2-D signal is used as input, whereas for the operations of the succeeding levels of decomposition, the decimated LL subband signal from the previous resolution level is used as input.

4.1.1 Formulation for the Computation of Four Subbands

Let a 2-D signal be represented by an $N_0 \times N_0$ matrix $\mathbf{S}^{(0)}$, with its (m,n)th element denoted by $S^{(0)}(m,n)$ $(0 \le m,n \le N_0-1)$, where N_0 is chosen to be 2^J, J being an integer. Let the coefficients of a 2-D FIR filter P (P=HH, HL, LH, LL) be represented by an $L \times M$ matrix $\mathbf{H}^{(P)}$. The (k,i)th coefficient of the filter P is denoted by $H^{(P)}(k,i)$ $(0 \le k \le L-1$, $0 \le i \le M-1)$. The decomposition at a given level $j=1, 2, \cdots, J$ can be expressed as

$$A^{(j)}(m,n) = \sum_{k=0}^{L-1} \sum_{i=0}^{M-1} H^{(\mathrm{HH})}(k,i) \cdot S^{(j-1)}(2m-k, 2n-i) \tag{4.1a}$$

$$B^{(j)}(m,n) = \sum_{k=0}^{L-1} \sum_{i=0}^{M-1} H^{(\mathrm{HL})}(k,i) \cdot S^{(j-1)}(2m-k, 2n-i) \tag{4.1b}$$

$$D^{(j)}(m,n) = \sum_{k=0}^{L-1} \sum_{i=0}^{M-1} H^{(\mathrm{LH})}(k,i) \cdot S^{(j-1)}(2m-k, 2n-i) \tag{4.1c}$$

$$S^{(j)}(m,n) = \sum_{k=0}^{L-1} \sum_{i=0}^{M-1} H^{(\mathrm{LL})}(k,i) \cdot S^{(j-1)}(2m-k, 2n-i) \tag{4.1d}$$

where $A^{(j)}(m,n)$, $B^{(j)}(m,n)$, $D^{(j)}(m,n)$ and $S^{(j)}(m,n)$ $(0 \le m,n \le N_j-1)$ denote the (m,n)th elements of the four $N_j \times N_j$ ($N_j = N_0/2^j$) matrices, $\mathbf{A}^{(j)}$, $\mathbf{B}^{(j)}$, $\mathbf{D}^{(j)}$ and $\mathbf{S}^{(j)}$, respectively, representing the HH, HL, LH and LL subbands of the 2-D input signal at the jth level. It is seen from (4.1) that the four decomposed subbands at a level are obtained by performing four 2-D convolutions. Each 2-D convolution can be seen as a sum of the products of the $L \times M$ filter coefficients and the elements contained in an $L \times M$ window

sliding on a 2-D data. The decimation by a factor of two in both the horizontal and vertical dimensions can be accomplished by sliding the $L\times M$ window by two positions horizontally and vertically for the computation of two successive samples. Only the LL subband data of decomposition are used as input for the decomposition at the next level. After J iterations, the 2-D signal $\mathbf{S}^{(0)}$ is transformed into J resolution levels, with HH, HL and LH subbands from each of the first $J-1$ levels and HH, HL, LH and LL subbands from the last (Jth) level. Since $N_j=N_0/2^j$, the number of samples that need to be processed at each level j is one quarter of that at the preceding level.

4.1.2 Formulation for a Four-Channel Filtering Operation

In order to facilitate parallel processing for the 2–D DWT computation, the $L\times M$ filtering operation needs to be divided into multi-channel operations, each channel processing one part of the 2-D data. It is seen from (4.1) that the even and odd indexed elements are always operated on the even and odd indexed filter coefficients, respectively. The matrix $\mathbf{S}^{(j)}$ representing the LL subband at the jth level can, therefore, be divided into four $(N_j/2+L/2)\times(N_j/2+M/2)$ sub-matrices, $\mathbf{S}_{ee}^{(j)}$,$\mathbf{S}_{oe}^{(j)}$,$\mathbf{S}_{eo}^{(j)}$ and $\mathbf{S}_{oo}^{(j)}$, whose (m,n)th ($0\le m\le N_j/2+L/2-1$, $0\le n\le N_j/2+M/2-1$) elements are given by

$$\begin{aligned} S_{ee}^{(j)}(m,n) &= S^{(j)}(2m,2n) \\ S_{oe}^{(j)}(m,n) &= S^{(j)}(2m+1,2n) \\ S_{eo}^{(j)}(m,n) &= S^{(j)}(2m,2n+1) \\ S_{oo}^{(j)}(m,n) &= S^{(j)}(2m+1,2n+1) \end{aligned} \tag{4.2}$$

taking into consideration the periodic padding samples at the boundary [134]. It is seen from (4.2) that the data at any resolution level are divided into four channels for processing by first separating the even and odd indexed rows of $\mathbf{S}^{(j)}$, and then separating

the even and odd indexed columns of the resulting two sub-matrices. The data in each channel can then be computed by an (L/2×M/2)-tap filtering operation. In order to facilitate such a 4-channel filtering operation, the filter coefficients, as used in (4.1), need to be decomposed appropriately. Accordingly, the matrix $\mathbf{H}^{(P)}$ needs to be decomposed into four (L/2×M/2) sub-matrices, $\mathbf{H}_{ee}^{(P)}, \mathbf{H}_{oe}^{(P)}, \mathbf{H}_{eo}^{(P)}$ and $\mathbf{H}_{oo}^{(P)}$, whose (k,i)th ($0 \le k \le L/2-1$, $0 \le i \le M/2-1$) elements are given by

$$\begin{aligned} H_{ee}^{(P)}(k,i) &= H^{(P)}(2k,2i) \\ H_{oe}^{(P)}(k,i) &= H^{(P)}(2k+1,2i) \\ H_{eo}^{(P)}(k,i) &= H^{(P)}(2k,2i+1) \\ H_{oo}^{(P)}(k,i) &= H^{(P)}(2k+1,2i+1) \end{aligned} \tag{4.3}$$

respectively. By using (4.2) and (4.3) in (4.1), any of the four subband signals, $\mathbf{A}^{(j)}$, $\mathbf{B}^{(j)}$, $\mathbf{C}^{(j)}$ and $\mathbf{S}^{(j)}$, at the jth resolution level, can be computed as a sum of four convolutions using (L/2×M/2)-tap filters. For example, the LL subband given by (4.1d) can now be expressed as

$$\begin{aligned} S^{(j)}(m,n) &= \sum_{k=0}^{L/2-1}\sum_{i=0}^{M/2-1} H_{ee}^{(\mathrm{LL})}(k,i)\cdot S_{ee}^{(j-1)}(m+k,n+i) \\ &+ \sum_{k=0}^{L/2-1}\sum_{i=0}^{M/2-1} H_{oe}^{(\mathrm{LL})}(k,i)\cdot S_{oe}^{(j-1)}(m+k,n+i) \\ &+ \sum_{k=0}^{L/2-1}\sum_{i=0}^{M/2-1} H_{eo}^{(\mathrm{LL})}(k,i)\cdot S_{eo}^{(j-1)}(m+k,n+i) \\ &+ \sum_{k=0}^{L/2-1}\sum_{i=0}^{M/2-1} H_{oo}^{(\mathrm{LL})}(k,i)\cdot S_{oo}^{(j-1)}(m+k,n+i) \end{aligned} \tag{4.4}$$

At any resolution level, the separation of the subband processing corresponding to even and odd indexed data as given by (4.4) is consistent with the requirement of decimation of the data in each dimension by a factor of two in the DWT computation. It is also seen from (4.4) that the filtering operations in the four channels are independent and identical, which can be exploited in the design of an efficient pipeline architecture for the 2-D DWT computation.

4.2 Pipeline for the 2-D DWT Computation

In a pipeline structure for the DWT computation, multiple stages are used to carry out the computations of the various resolution levels of the transform [144]. The computation corresponding to each resolution level needs to be mapped to a stage or stages of the pipeline. It is seen from the formulation in Section 4.1 that the task of computing the jth resolution level in a J-level DWT computation consists of computing $N_0^2/4^{j-1}$ samples, where $N_0=2^J$. The computation of each sample actually performs an ($L\times M$)-tap HH, HL, LH or LL FIR filtering operation that comprises the operations of ($L\times M$) multiplications followed by ($L\times M$) accumulations. Assuming that these operations for the computation of one sample are carried out by a unit of filter processor, the overall task of the DWT computation would require a certain number of such filter units. In order to design a pipeline structure capable of performing a fast computation of the DWT with low expense on hardware resources and low design complexity, an optimal mapping of the overall task of the DWT computation to the various stages of the pipeline needs to be determined. Any distribution of the overall task of the DWT computation to stages must consider the inherent nature of the sequential computations of the resolution levels that limit the computational parallelism of the pipeline stages, and consequently the latency of the pipeline. The key factors in the distribution of the task to the stages are the maximization of the inter-stage and intra-stage computational parallelism and the synchronization of the stages within the constraint of the sequential nature of the computation of the resolution levels. The feature of identical operations associated with the computations of all the output samples irrespective of the resolution levels in a DWT computation can be exploited to maximize the intra-stage parallelism of the pipeline.

Further, in order to minimize the expense on the hardware resources of the pipeline, the number of filter units used by each stage ought to be minimum and proportional to the amount of the task assigned to the stage.

A straightforward mapping of the overall task of the DWT computation to a pipeline is one-level to one-stage mapping, in which the tasks of J resolution levels are distributed to J stages of the pipeline. In this mapping, the amount of hardware resources used by a stage should be one-quarter of that used by the preceding stage. Thus, the ratio λ of the hardware resource used by the last stage to that used by the first stage has a value of $1/4^{J-1}$. For images of typical size, this parameter would assume a very small value. Hence, for a structure of the pipeline that uses identical filter units, the number of these filter units would be very large. Further, since the number of such filter units employed by the stages would decrease exponentially from one stage to the next in the pipeline, it will make their synchronization very difficult. The solution to such a difficult synchronization problem, in general, requires more control units, multiplexers and registers, which result in a higher design complexity. A reasonably large value of $\lambda<1$ would be more attractive for synchronization. In this respect, the parameter λ can be seen as a measure of design difficulty, with a smaller value of this parameter representing a greater design complexity.

The parameter λ can be increased from its value of $1/4^{J-1}$ in the one-level to one-stage pipeline structure by dividing the large-size stages into a number of smaller stages or merging the small-size stages into larger ones. However, dividing a stage of the one-level to one-stage pipeline into multiple stages would require a division of the task associated with the corresponding resolution level into sub-tasks, which in turn, would

call for a solution of even a more complex problem of synchronization of the sub-tasks associated with divided stages. On the other hand, merging multiple small-size stages of the pipeline into one stage would not create any additional synchronization problem. As a matter of fact, such a merger could be used to reduce the overall number of filter units of the pipeline.

In view of the above discussion, the synchronization parameter λ can be increased by merging a number of stages at tail end of the pipeline. Fig. 4.1 shows the structure of a pipeline in which the stages I to J of the one-level to one-stage pipeline have been merged. In this structure, the tasks of the resolution level from $j=1$ to $j=I-1$ are mapped to stage 1 to $I-1$, respectively, whereas those of the resolution levels $j=I, \cdots, J$, are mapped all together to the Ith stage. Note that the total amount of computations performed by stage I is less than one-half of that performed by stage $I-1$. Considering the fact that the number of filter units employed by each stage of the pipeline is an integer, it is reasonable to have the ratio of the numbers of filter units used by the last two stages (i.e., stages $I-1$ and I) to be 2:1. The value of the parameter λ is now increased from $1/4^{J-1}$ to $1/4^{I-1.5}$. However, now the resources employed by stage I would not be fully utilized, which would lower the efficiency of the hardware utilization of the pipeline of Fig. 4.1. Assume that the parameter η represents the hardware utilization efficiency defined as the ratio of the resources used to that employed by the pipeline. The hardware utilization efficiency η of the pipeline in Fig. 4.1 can be shown to be equal to $(1-4^{-J})/(1+4^{-I+0.5})$. Since for images of typical size, 4^{-J} is negligibly small compared to one, the expression for η can be simplified as $1/(1+4^{-I+0.5})$. As the number of stages I employed by the pipeline increases, the hardware utilization efficiency increases with the

parameter η approaching unity for a maximum efficiency. On the other hand, the difficulty in synchronizing the stages gets worse as the parameter λ decreases with increasing value of *I*. A variation in the value of *I* results in the values of λ and η that are in conflict from the point of view of stage synchronization and hardware utilization efficiency. Therefore, a value of *I* needs to be determined that optimizes the values of λ and η jointly.

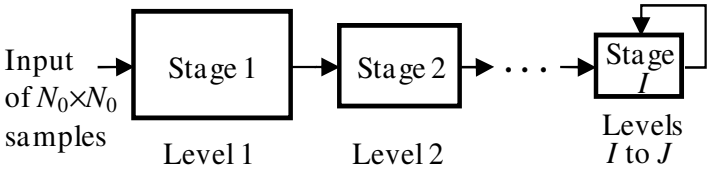
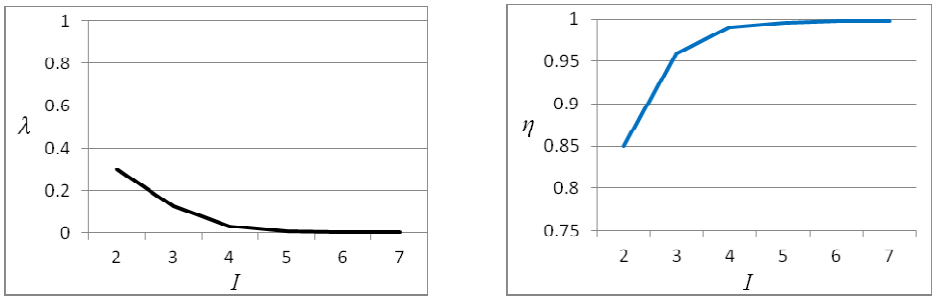


Figure 4.1: Pipeline structure with *I* stages for *J*-level computation.

Considering an example of an image of size $2^8 \times 2^8$, in which case $J=8$. Fig. 4.2 gives the plots of λ and η as a function *I*, the number of stages employed by the pipeline. It is seen from this figure that $I=3$ provides the best compromise between the values of λ and η. Therefore, a 3-stage pipeline with an acceptable value for the synchronization parameter and high hardware utilization efficiency would be the best choice of a pipeline. Note that the size of the images used in typical applications would have little bearing on the conclusion thus reached regarding the number of stages employed in the pipeline. Also, note that a 3-stage pipeline can perform the DWT computation for a variable number of resolution levels from 3 to *J*. With three as the optimal choice of the number of stages in a pipeline, one can now choose the minimum numbers of filter units as 8, 2 and 1 for the stages 1, 2 and 3 in order to perform the tasks associated with the resolution levels 1, 2 and 3 to *J* together, respectively. The next section is concerned specifically with a detailed design of the 3-stage pipeline structure.



(a) (b)

Figure 4.2: Parameters λ and η plotted as functions of the number of stages *I* used in a pipeline architecture. (a) λ versus *I*, (b) η versus *I*.

4.3 Design of the Architecture

In the previous section, we advocated a three-stage pipeline structure for the computation of the 2-D DWT to realize an optimal combination of the parameters for the hardware utilization and pipeline synchronization. In this three-stage structure, like in any pipeline architecture, the operations in a given stage depend on the data produced by the preceding stage. However, because of the way that the computational load of the various resolution levels of the 2-D DWT computation has been distributed among the three stages, the operations in the first and second stages of the pipeline do not depend on the data produced by themselves, whereas that in stage 3 does depend on the data produced by itself. The operations of the three stages need to be synchronized in a manner so that the three stages perform the computation of multiple resolution levels within a minimum possible time period while using the available hardware resources maximally. In this section, we present the design of the proposed 3-stage pipeline architecture, starting with the synchronization of the operations of the stages, and then focusing on the details of the intra-stage design so as to provide an optimal performance.

4.3.1 Synchronization of Stages

Recall from Section 4.2 that the distribution of the computational load among the three stages, and the hardware resources made available to them are in the ratio 8:2:1. Accordingly, the synchronization of the operations between the stages needs to be carried out under this constraint of the distribution of the computational load and hardware resources. According to the nature of the DWT, the computation of a resolution level j depends on the data computed at its previous level $j-1$, in which the number of computations is four times of that at the resolution level j. Therefore, the stages of pipeline need to be synchronized in such a way that each stage starts the operation at an earliest possible time when the required data become available for its operation. Once the operation of a stage is started, it must continue until the task assigned to it is fully completed.

Consider the timing diagram given in Fig. 4.3 for the operations of the three stages, where t_1, t_2 and t_3 are the times taken individually by stages 1, 2 and 3, respectively, to complete their assigned tasks, and t_a and t_b are the times elapsed between the starting points of the tasks by stages 1 and 2, and that by stages 2 and 3, respectively. Note that the lengths of the times t_1, t_2 and t_3 to complete the tasks by individual stages are approximately the same, since the ratios of the tasks assigned and the resources made available to the three stages are the same. The average times to compute one output sample by stages 1, 2 and 3 are in the ratio 1:4:8. In Fig. 4.3, the relative widths of the slots in the three stages are shown to reflect this ratio. Our objective is to minimize the total computation time $t_a+t_b+t_3$ by minimizing t_a, t_b and t_3 individually.

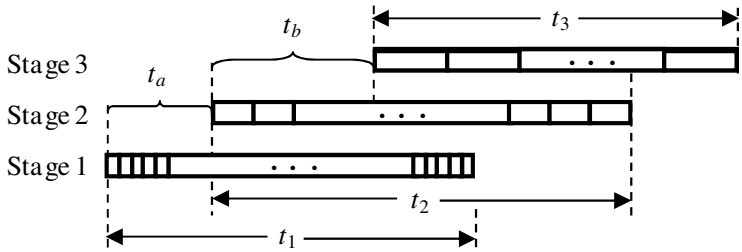


Figure 4.3: Timing diagram for the operations of three stages.

Assume that 2-D output samples for a resolution level are computed row-by-row starting from the upper-left corner sample. Since the operations in stage 1 are independent of those in the other two stages, it can operate continuously to compute all the samples of level 1. The value of t_1 is equal to $T_s N_1^2$, where T_s is the average time taken by stage 1 to compute one output sample. Since the operations of stages 2 and 3 require the output data computed by stages 1 and 2, respectively, their operations must be delayed by certain amount of times so that they can operate continuously with the data required by them becoming available. We now give the lowest bound on t_a and t_b so that once stages 2 and 3 start their operations they could continue their operations uninterruptedly. Since the operation of stage 2 starts at time t_a, the (i,k)th output sample of level 2, denoted by $S^{(2)}(i,k)$, will be computed starting at the time instant $t_x=t_a+4T_s(i\cdot N_2+k)$, where $4T_s$ is the average time taken by stage 2 to compute one output sample. Using (4.1), among the level-1 samples required for the computation of $S^{(2)}(i,k)$, the $(2i+L-1, 2k+M-1)$th level-1 sample, denoted by $S^{(1)}(2i+L-1, 2k+M-1)$, is the latest output sample computed at the time instant $t_y=T_s[N_1(2i+L-1)+2k+M-1]+T_s$. Now, if at the time of starting the calculation of the output sample $S^{(2)}(i,k)$, i.e. t_x, the sample $S^{(1)}(2i+L-1,2k+M-1)$ has already been calculated by stage 1, all the leve-1 samples necessary to calculate this level-2 output sample would be available. This requires us to

impose the constraint $t_x > t_y$, for all i and k, i.e. $0 \le i,\ k \le N_2 - 1$. This condition implies that

$$t_a > T_s(N_1 L - N_1 + M - 2k) \tag{4.5}$$

The minimum value of t_a is given by

$$t_{a\min} = T_s[N_1(L-1) + M] \tag{4.6}$$

Assume that stage 3 computes all the output samples of all remaining levels (i.e. level 3 to level J) in a sequential manner. We only need to consider the requirement of the data availability for the computation of level-3, which uses the level-2 samples computed by stage 2. Then, in a way similar to that obtaining $t_{a\min}$, by imposing the condition that at the time instant of starting the calculation of a level-3 output sample by stage 3, all the samples in the window of the level-2 output samples are available, it can be shown that the minimum value of t_b is given by

$$t_{b\min} = 4T_s[N_2(N_2/2 + L - 2) + M] \tag{4.7}$$

Based on the above discussion, the operations of the three stages can be arranged in the following manner:

Step 1. Stage 1 operates continuously on the input signal to compute the level-1 output samples sequentially.

Step 2. Stage 2 starts its operation immediately following the computation of the ($L-1$, M)th level-1 output sample, $S^{(1)}(L-1, M)$, and then continues its operation of all other level-2 output samples in a sequential manner.

Step 3. Stage 3 starts its operation for the computation of level-3 samples immediately after stage 2 completes the computation of the ($N_2/2+L-2$, $M-1$)th level-2 output sample, $S^{(2)}(N_2/2+L-2, M-1)$, and then continues the computation of other level-3 output samples sequentially. Computations of the output samples of levels 4 to J are

carried out sequentially by the stage 3 following the computation of level-3 output samples.

4.3.2 Design of Stages

As discussed in Section 1.4, in the proposed three-stage architecture, stages 1 and 2 perform the computations of levels 1 and 2, respectively, and stage 3 that of all the remaining levels. Since the basic operation of computing each output sample, regardless of the resolution level or the subband, is the same, the computation blocks in the three stages can differ only in the number of identical processing units employed by them depending on the amount of the computations assigned to the stages. As seen from (4), an ($L\times M$)-tap filtering operation is decomposed into four independent ($L/2\times M/2$)-tap filtering operations, each operating on the 2-D $L/2\times M/2$ data resulting from the even or odd numbered rows and even or odd numbered columns of an $L\times M$ window of an LL-subband data. A unit consisting of $L/2\times M/2$ MAC cells can now be regarded as the basic *processing unit* to carry out an ($L/2\times M/2$)-tap filtering operation. An $L\times M$ window of the raw 2-D input data or that of an LL-subband data must be decomposed into four distinct $L/2\times M/2$ sub-windows in accordance with the four decomposed terms given by the right side of (4). This decomposition of the data in an $L\times M$ window can be accomplished by designing for each stage an appropriate data scanning unit (DSU) based on the way the raw input or the LL-subband data is scanned. The stages would also require memory space (buffer) to store the raw input data or the LL-subband data prior to scanning. Since stages 1 and 2 need to store only part of a few rows of raw input or LL-subband data at a time, they require a buffer of size of O(N), whereas since stage 3 needs to store the entire

LL-subband data of a single resolution level, it has a buffer of size of O(N^2). Fig. 4.4 gives the block diagram of the pipeline showing all the components required by the three stages. Note that the data flow shown in this figure comprises only the LL-subband data necessary for the operations of the stages. The HH, HL and LH subband data are outputted directly to an external memory. Now, we give details on the structure of the data scanning unit to scan the 2-D data and establish four distinct $L/2 \times M/2$ sub-windows, as well as on the distribution of the filtering operations to the processing units in each stage.

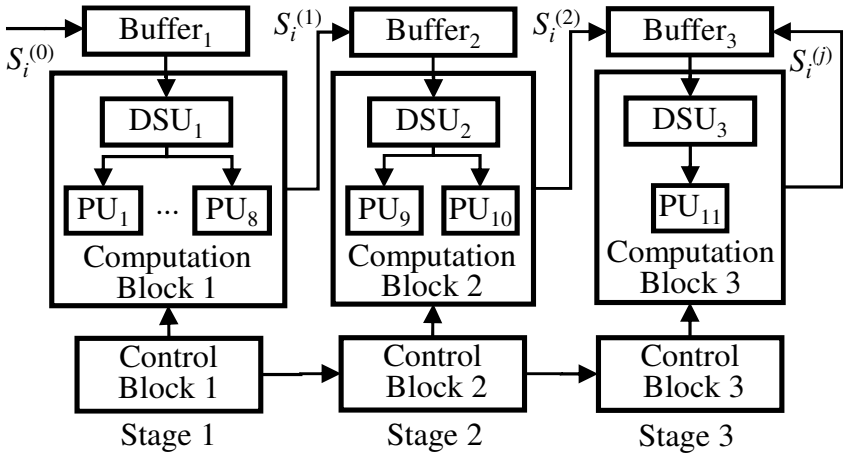


Figure 4.4: Block diagram of the three-stage architecture.

(a) Structure of the Data Scanning Unit

In accordance with (4.4), an $L \times M$ window of the raw 2-D input data stored in Buffer_1 or an LL-subband data stored in Buffer_2 or Buffer_3 must be partitioned into four $L/2 \times M/2$ sub-windows, and stored into the DSU of the corresponding stage. Further, this same equation also dictates that a 2-D input data must be scanned in a sequential manner shown in Fig. 4.5(a). According to this sequence of scanning, the samples in a set of data

comprising L rows of a 2-D input data are scanned starting from the top-left corner. Once the scanning of all the samples of L rows is completed, the process is repeated for another L rows after shifting down by two row positions. The objective is then to design a structure for a DSU so that samples scanned with this sequential mode get partitioned into the four sub-windows (Fig. 4.5(b)).

In order to partition an $L\times M$ window into four $L/2\times M/2$ sub-windows, the structure of the DSU must first partition the samples of the window into two parts depending on whether a sample belongs to an even-indexed or odd-indexed row; then the samples in each part must be partitioned further into two parts depending on whether a sample belongs to an even-indexed or odd-indexed column. The first partition can be achieved by directing scanned samples alternatively to two sets of $L/2$ shift registers. The second partition can be achieved by reorganizing the samples stored in the shift registers of the two sets depending on whether a sample belongs to even-indexed or odd-indexed column

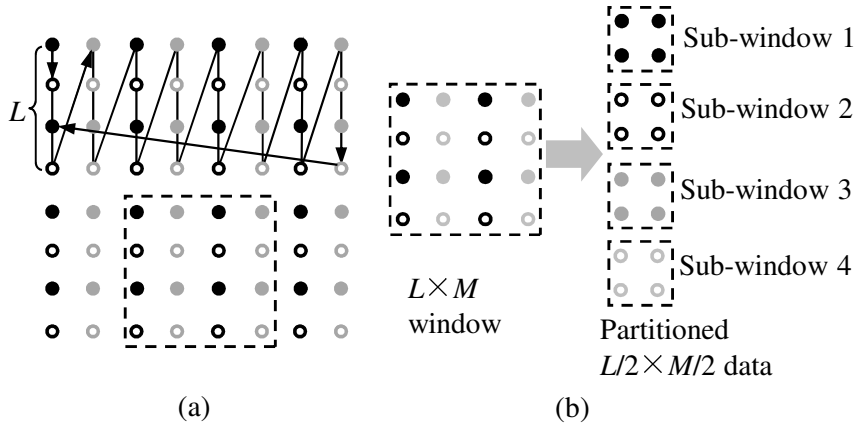


Figure 4.5: Diagram illustrating the data scanning. (a) Scanning of an $N_j\times N_j$ 2-D data. (b) Partitioning of an $L\times M$ window into four $L/2\times M/2$ sub-windows. The solid and empty circles represent the samples in even-indexed and odd-indexed rows, respectively, whereas the black and grey circles represent the samples in even-indexed and odd-indexed columns, respectively.

by employing demultiplexers. Finally, the samples of the four sub-windows can be stored, respectively, into four units of $L/2 \times M/2$ parallel registers. Fig. 4.6 shows a structure of the DSU to accomplish this task. This data scanning scheme automatically incorporates the downsampling operations by two in the vertical and horizontal directions (as required by the transform), and thus no additional peripheral circuits and registers are required for the downsampling operations by the architecture. As a result, the data scanning scheme, in comparison to the other schemes [145], requires less hardware resources for the control units and fewer registers for the stages.

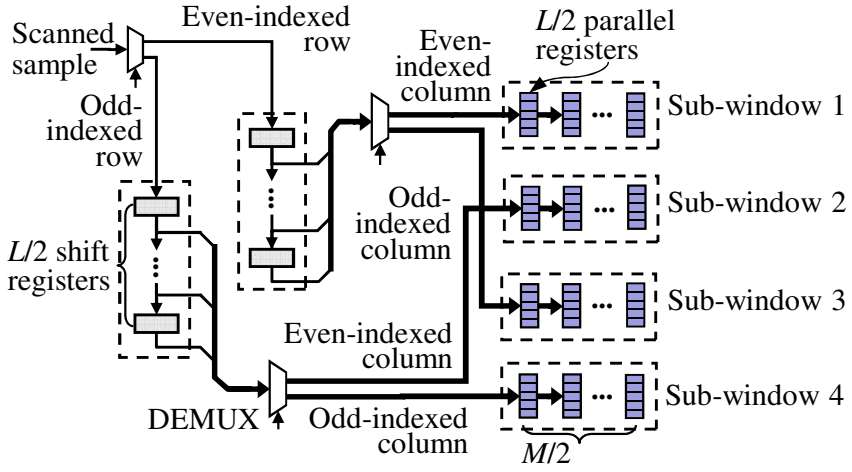


Figure 4.6: Structure of the data scanning unit (DSU).

(b) Distribution of filtering operations among the processing units employed by stages

In accordance with (4.1) and (4.4), decomposing input data into four subbands requires four $L \times M$ filtering operations, and each of the four filtering operations requires four $(L/2 \times M/2)$-tap filtering operations. Thus, a total of 16 $(L/2 \times M/2)$-tap filtering operations are involved for the computation of the samples for the four subbands using an

$L \times M$ window of the input data. Now, for each stage, these 16 types of filtering operations must be assigned to the processing units available to the stage using four sub-windows of data from its DSU. Given the available resources of the stages, the objective here is to process the 16 types of filtering operations with maximized computational parallelism and with priority given to the computation of the samples of LL subband.

In stage 1, since eight processing units are available, the processing task can be distributed among them so that one processing unit carries out the subtask of $(L/2 \times M/2)$-tap filtering operations corresponding to a pair of subbands from the LL, LH, HL and HH using the data of one sub-window. One such distribution of the task is shown in Fig. 4.7, from which it is seen that each of the processing units PU_1 to PU_4 carries out the LL and LH filtering operations sequentially using the sub-windows 1 to 4, respectively, whereas each of the processing units PU_5 to PU_8 carries out the HH and HL filtering operations using the same sub-windows. In stage 1, the LL and HH subband samples are produced in parallel in one clock cycle, whereas the LH and HL subband samples are produced in parallel in the next.

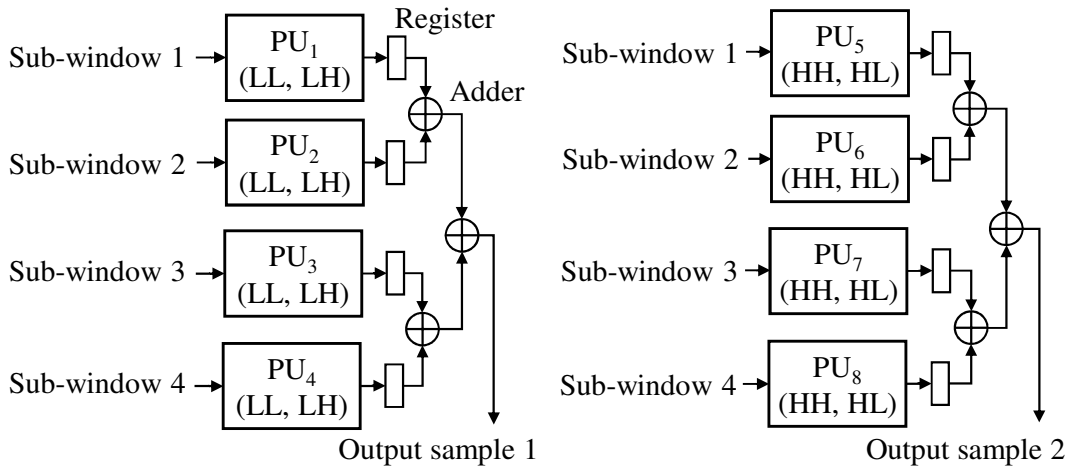


Figure 4.7: Structure of eight processing units employed by stage 1.

Since stage 2 employs two processing units, each must perform the task of all the four subbands using two sub-windows. As the data of the four sub-windows, 1 to 4, become available in a sequential manner, sub-windows 1 and 3 are sequentially assigned to PU_9, whereas sub-windows 2 and 4 in a similar manner are assigned to PU_{10}. This distribution of the task for stage 2 is shown in Fig. 4.8, from which it is seen that each of the processing units, PU_9 and PU_{10}, carries out the $(L/2 \times M/2)$-tap filtering operations. In stage 2, PU_9 and PU_{10} operating in parallel produce the LL, LH, HH and HL subband samples sequentially in eight consecutive clock cycles.

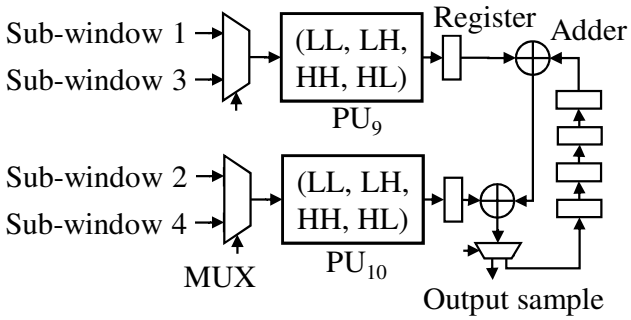


Figure 4.8: Structure of two processing units employed by stage 2.

Since only one processing unit, PU_{11}, is employed by stage 3, it has to carry out all the filtering operations for each of the four sub-windows, as shown in Fig. 4.9. In this figure, the four sub-windows, 1 to 4, are chosen successively, as input to PU_{11}. For each sub-window, the processing unit PU_{11} then carries out the $(L/2 \times M/2)$-tap filtering operations. In this stage, PU_{11} produces sequentially the LL, LH, HH and HL subband samples in 16 consecutive clock cycles.

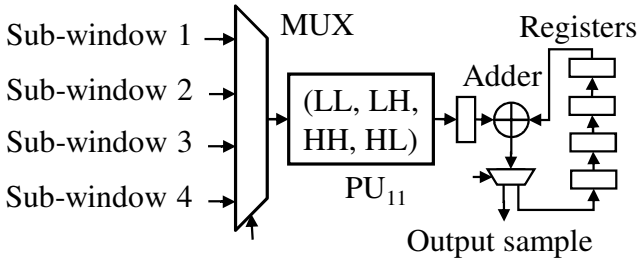


Figure 4.9: Structure of one processing unit employed by stage 3.

Note that one processing unit at a time processes the samples of only one sub-window corresponding to one of the four subbands. Assume that such a processing time by a processing unit to be one time unit. Now, since stages 1, 2 and 3 have 8, 2 and 1 processing units, respectively, they can process sub-windows at the rates of 2, 1/2 and 1/4 sub-windows per unit time. This coupled with the fact that the processing loads (i.e. the number of sub-windows) assigned to the three stages are in the ratio 8:2:1, lets us to conclude that the operations of the three stages are mutually synchronized.

(c) Design of the Processing Unit

In each stage, a processing unit carries out an ($L/2 \times M/2$)-tap filtering operation using the samples of an $L/2 \times M/2$ sub-window at a time to produce the corresponding output. Since the sub-windows cannot be fed into a processing unit at a rate faster than the rate at which these sub-windows are processed by the processing unit, the processing time to process a sub-window (one time unit) is critical in determining the maximum clock frequency at which the processing units can operate. Each physical link from a given bit of the input to an output bit of the processing unit gives rise to a data path having a delay that depends on the number and the types of operations being carried out along that path.

Therefore, it is crucial to aim at achieving the shortest possible delay for the critical path when designing a processing unit for our architecture [131]–[133], [142], [143].

The filtering operation carried out by a processing unit, as described above, can be seen as $L/2 \times M/2$ parallel multiplications followed by an accumulation of the $L/2 \times M/2$ products. If the input samples and the filter coefficients have the wordlengths of X and Y bits, respectively, then the processing unit produces an array of $(Y*L*M/4) \times X$ bits simultaneously in one clock cycle.

In order to obtain the output sample corresponding to a given sub-window, the bits of the partial products must be accumulated vertically downward and from right to left by taking the propagation of the carry bits into consideration. The task of this accumulation can be divided into a sequence of layers. The shortest critical data path can be achieved by minimizing the number of layers and the delay of the layers. In each layer, a number of bits consisting of the partial product bits and/or the carry bits from different rows need to be added. This can be done by employing in parallel as many bit-wise adders as needed in each layer. The idea behind using bit-wise adder is to produce to the extent possible the number of output bits from a layer is smaller than the number of input bits to that layer. This can be done by using full adders and specifically designed double adders, in which the full adder consumes 3 bits and produces 2 bits (one sum and one carry bits) whereas the double adder consumes two pairs of bits (2×2) from neighbouring columns and produces 3 bits (one sum and two carry bits/two sum and one carry bits). The two types of adders have equal delay, and are efficient in generating carry bits and compressing the number of partial products [133]. With this structure of the layers, the number of layers becomes minimum possible and the delay of a layer is equal to that of a

full adder or equivalently to that of a double adder, thereby providing the shortest critical path for the accumulation network.

Since the two rows of bits produced by the accumulation network still remain unaccumulated, they finally need to be added to produce one row of output bits in the final phase of the task of a processing unit by using a carry propagation adder. Note that tasks of the accumulation network and the carry propagation adder can be made to have some partial overlap, since the latter can start its processing as soon as the rightmost pairs of bits becomes available from the former. Fig. 4.10 depicts a block diagram of a processing unit based on the above discussion.

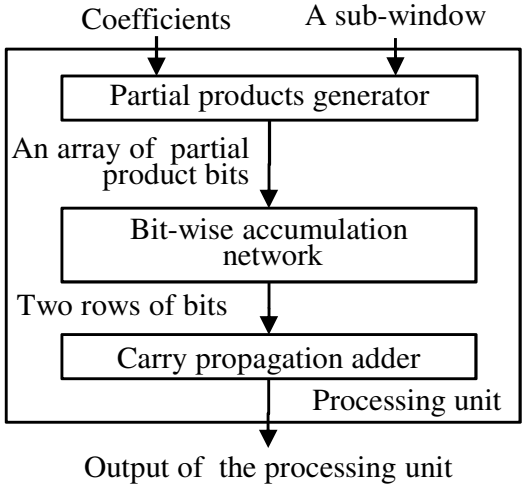


Figure 4.10: Block diagram of a processing unit.

4.4 Performance Results and Comparisons

4.4.1 Performance of the Proposed Architecture

In order to evaluate the performance of a computational architecture, one needs to make use of certain metrics that characterize the architecture in terms of the hardware resources used and the computation time. In this chapter, the hardware resources used for the filtering operation are measured by the number of multipliers (N_{MUL}) and the number of adders (N_{ADD}), and that used for the storage of data and filter coefficients are measured by the number of registers (N_{REG}). The computation time, in general, is technology dependent. However, a metric that is technology independent and can be used to determine the computation time T is the number of clock cycles (N_{CLK}) elapsed between the first and the last samples inputted to the architecture. Assuming that one clock period is T_c, the total computation time can then be obtained as $T=N_{CLK}T_c$.

For a J-level 2-D DWT computation of an $N\times N$ image using $(L\times L)$-tap filters, the expressions for the metrics mentioned above for the proposed 3-stage architecture are given in Table 4.1. It is seen from this table that the numbers of multipliers, adders and registers in the DSUs employed by the architecture depend only on the filter length, whereas the number of the registers of the buffers depends also on the image size.

Table 4.1: Performance metrics for the proposed 2-D architecture

N_{CLK}	N_{MUL}	N_{ADD}	N_{REG}	
			DSUs	Buffers
$N^2/2$	$11L^2/4$	$11\text{Log}_2(L^2/2)+9$	$3L^2+3L$	$3NL/4+3N^2/128$

In order to evaluate the performance of the proposed architecture in terms of T_c, we consider an example of designing a circuit for the DWT computation of an image of size N=512. For this purpose, we use 2-D filters of size L=M=4, wordlength for the filter coefficients as 8-bit, and the number of resolution levels J=6. The input samples are encoded by using a radix-4 booth encoder and used as one of the two operands for the multiplication operation. All the carry propagation adders of the architecture have a 16-bit wordlength and use a structure that combines the carry-skip and carry-select adders [135]. The circuit is synthesized in RTL by using Synopsys with 0.18-µm ***CMOS*** technology. The synthesized results show that the circuit can operate with a minimum clock period of 6.5 ns (i.e. at a maximum clock frequency of 153 MHz). The circuit has a core area of 4.95×3.84 mm^2, and consists of 850K logic gates and a 24.5K-RAM. The power consumed by the circuit is obtained as 214 mW at 100 MHz clock frequency.

Table 4.2: Resources utilized in FPGA device for the circuit implementation for the 2-D DWT computation when N=512, L=M=4 and J=6

Resource	Number used	Percentage used
CLB Slices	2842	20%
Flip-flop Slices	1059	3%
4-input LUTs	4989	18%
Bonded IOBs	130	23%
BRAMs	8	5%

In order to validate the circuit design based on the proposed architecture, the circuit is implemented on a typical FPGA board, Virtex-II Pro XC2VP30-7. The board is capable of operating with a clock frequency of up to 400 MHz at a core voltage of V_{DD}=1.5 V. The resources utilized by the FPGA implementation in terms of the numbers of configuration logic block (CLB) slices, flip-flop slices, 4-input look-up tables (LUTs), input/output blocks (IOBs) and block RAMs (BRAMs) are given in Table 4.2. The circuit implemented is found to perform well with a clock period as short as 7.4 ns (i.e. a maximum clock frequency of 134 MHz). The time for the DWT computation of an image of size 512×512 is 0.97 ms. In other words, the circuit is able to process motion pictures with a speed of 1022 frames per second (FPS). The power consumption of the FPGA device on which the circuit is implemented is measured to be 303 mW at 100 MHz clock frequency. This measured value for the power consumption compares reasonably well with the simulated value of 214 mW, considering that the measured value also includes the power dissipated by the unused slices within the FPGA device.

In order to validate the proposed architecture further, various circuits, which are designed based on the proposed architecture for the values of N=128, 256, 512, 1024, 2048 and J=3, 6, are implemented on the same type of FPGA board as used above. The implementation results for the various circuits are shown in Fig. 4.11. It is seen from this figure that the number of CLB slices (N_{CLB}) changes very slightly with the image size N or the number of resolution levels J (Fig. 4.11(a)), while the number of BRAMs (N_{BRAM}) increases rapidly (Fig. 4.11(b)). These results are consistent with the performance evaluation results provided in Table 4.1, and also demonstrate that the circuits for the DWT computation of images of different size and with different number of resolution

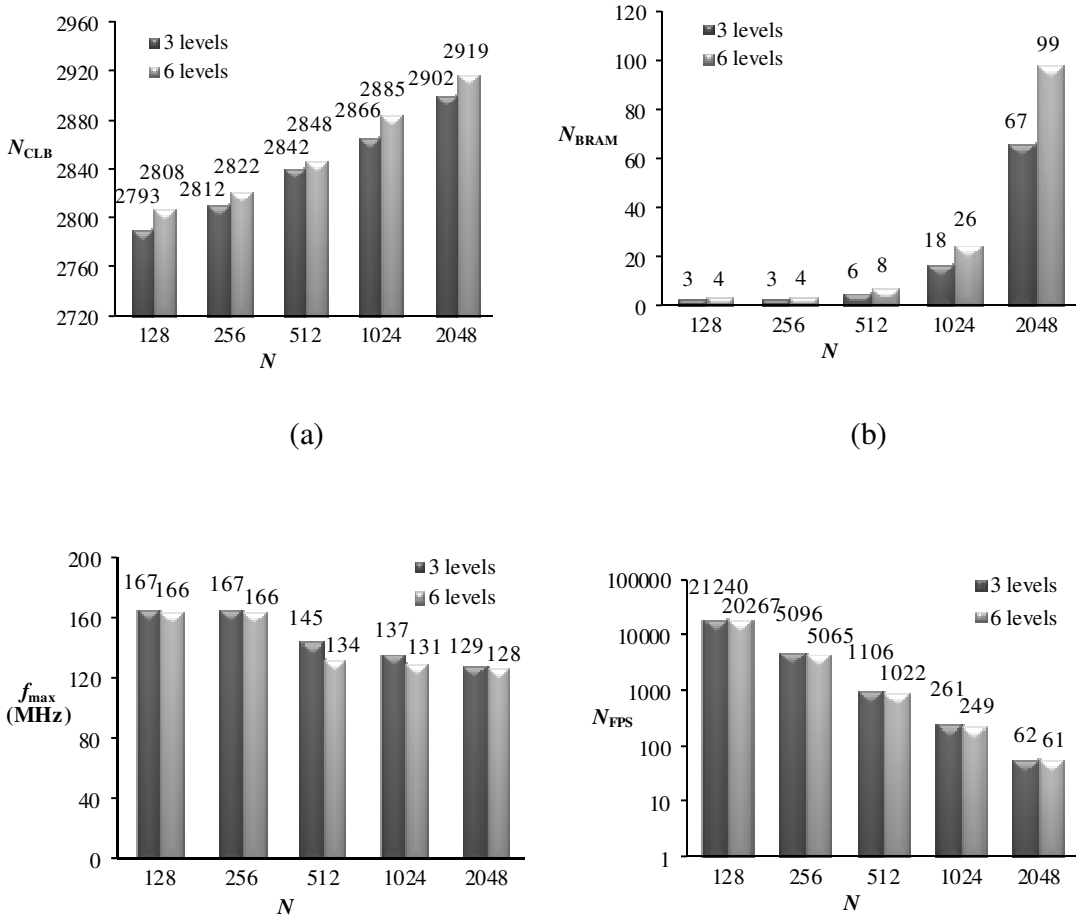


Figure 4.11: Results of various FPGA implementations with N=128, 256, 512, 1024, 2048, and J=3, 6. (a) The numbers of CLB slices versus N, (b) the numbers of BRAMs versus N, (c) the maximum clock frequencies versus N, and (d) the numbers of frames per second versus N.

levels can be implemented essentially by varying the size of the buffer used. The performance of only a slight decrease in the maximum clock frequency (f_{max}) and that of a logarithmic decrease in the number of frames per second (N_{FPS}), as the image size increases (Fig. 4.11(c) and (d)), are in conformity with the normal expectation.

4.4.2 Comparisons of Various 2-D Architectures

In order to compare the hardware utilization and computation time of the proposed and other architectures, expressions for the relevant performance metrics for a J-level DWT computation of an $N\times N$ image using $(L\times L)$-tap filters for the various architectures are given in Table 4.3. It is seen from this table that the architecture of Prop. 4 in [146] and that of [66], require, respectively, $N^2/12$ and $N^2/4$ clock cycles, which are smaller than $N^2/2$ clock cycles required by the proposed architecture. This performance of [66] is achieved by utilizing the hardware resources of adders and multipliers that is more than twice of that required by the proposed architecture. Also, it is to be noted that in [146] the amount of the hardware resources (adders, multipliers and delay units) is larger than that required by the proposed architecture. Indeed, a smaller value of N_{CLK} does not necessarily mean a smaller computation time T, since the clock period T_c may significantly differ from one architecture to another. It is also seen from Table 4.3 that the hardware utilization of the proposed architecture is higher than that of the pipeline architectures in [44], [66], [147] and [148], and it is only slightly lower than that of [146], in which 100% hardware utilization is achieved by using a much larger number of adders. Furthermore, the proposed architecture provides a shorter latency compared with the architectures in [44], [147], [149] and [150] that use 1-D type filters. On the other hand, the architectures in [146] and [66] provide smaller latencies, but employ proportionally larger hardware resources.

Table 4.3: Performance metrics for various 2-D architectures

Architecture	No. of multipliers	No. of adders	Storage size	Filter type	No. of clock cycles	Hardware utilization	Latency
Recursive architecture [44]	12	16	$4N$	1-D (9/7)	N^2+N	50%-70%	T_cN^2
Generic folded [149]	$6J$ $(L/2)$	$6J(1+\log_2(L/2))$	$4(L-1)N/3$	1-	N^2	N/A	T_cN^2
Symmetrically extended [147]	$L/2+L/4+L/8$	$2(L/2+L/4+L/8)$	$(L+0.5)N$	1-D	$1.5N^2$	87.5%	$1.5T_cN^2$
Parallel FDWT [150]	12	16	$3N/2$	1-D (9 7)	N^2	N/A	T_cN^2
Line-based [151]	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Parallel Prop. 4 [146]	96	240	$[4N+32J+256]$ (on chip delay units) $[8N+128(J-1)]$ (off chip buffer)	2-D $(L=4)$	$N^2/12$	100%	$T_cN^2/12$
Arch2D-II [152]	$L^2/2$	$L^2/2+L$	N/A	2-D	$2N^2/3$	N/A	$2T_cN^2/3$
Pipeline [66]	$6L^2$	$6L^2$	$2NL$	2-D	$N^2/4$	66.7%	$T_cN^2/4$
Parallel structure [148]	48	24	$6N/2+6N/4$ $(J=3)$	2-D (4×4)	$L^2N^2/16+L^2N/8$	5.6%	N/A
Proposed	$11L^2/4$	$11\log_2(L^2/2)+9$	$3L+3L^2$ (on chip delay units) $3NL/4+3N^2/128$ (off chip buffer)	2-D	$N^2/2$	96%	$T_cN^2/2$

N/A: Not available

The performance of the proposed architecture is now compared with various other architectures in terms of the FPGA implementation results available in the literature. The FPGA implementation results for the architectures presented in [44] and [147]–[152] are listed in Table 4.4. It is seen from this table that the implemented circuit for the proposed architecture requires a time of 0.97 ms to compute a 6-level DWT of an image of size 512×512, which is about one-half and one-third of the closest computation times offered by the implementations of the architectures of [152] and [150], respectively. In comparison to the architecture of [150], the proposed architecture provides this 3 times increase in the speed of computation at the expense only about 67% increase in the hardware. In comparison to the architecture of [152], the proposed architecture provides an improvement of 50% in the speed of computation while at the same times consumes about 35% less hardware resources. In order to have a fair comparison with the non-separable architecture of [152], whose computation time is next best to that of the proposed architecture, we have implemented the latter also on Virtex 2000E. The implementation of the proposed architecture on this device results in a computation time of 1.4 ms and in 3430 used CLB slices. Thus, with the architecture of [152] and the proposed architecture implemented on the same FPGA device, the latter gives a 17% gain in the computational speed and 21% reduction in the hardware resources. Overall, the area-time and area-$(\text{time})^2$ products of the proposed architecture have values that are, respectively, at least 33% and 78% smaller than those of the other architectures.

Table 4.4: Comparison of various FPGA implementations

Architecture	Image size (N)	No. of CLB slices	RAM size (bits)	f_{max} (MHz)	T (ms)	Area×T*	Area×T^2	Device
Recursive architecture [44]	512 (J=3)	879	10N	50	5.3	4659	24692	XC2V250
Generic folded [149]	256 (J=3)	4720	10×(4K)	75	0.874	4125	3605	Virtex 600E-8
Symmetrically extended [147]	512 (J=3)	2559	17×(18K)	44.1	9	23031	207279	XC2V500
Parallel FDWT [150]	512 (J=5)	1700	3N/2	171.8	3.1	5270	16337	Virtex 2
Line-based [151]	512 (J=6)	2950	4×(18K)	113.6	5.2	15340	79768	XC4VLX15
Parallel Prob. 4 [146]	Implementation results not available							
Arch2D-II [152]	512 (J=3)	4348	24×(18K)	105	1.7	7392	12566	Virtex 2000E
Pipeline [66]	Implementation results not available							
Parallel structure [148]	512 (J=3)	3580	2304	45	5.9	21122	124619	XCV600E
Proposed	512 (J=6)	2842	8×(18K)	135	0.97	2757	2674	XC2VP30

*The value of area in the calculation of area-time product is replaced by the No. of CLB slices since the former is proportional to the latter.

4.5 Summary

In this chapter, a scheme for the design of pipeline architectures for a high-speed non-separable computation of the 2-D DWT has been proposed. The objective has been to achieve a short computation time by maximizing the operational clock frequency ($1/T_c$) and minimizing the number of clock cycles (N_{CLK}) required for the 2-D DWT computation by developing a scheme for enhanced inter-stage and intra-stage computational parallelism for the pipeline architecture.

To enhance the inter-stage parallelism, a study has been undertaken that suggests that, in view of the nature of the DWT computation, it is most efficient to map the overall task of the DWT computation to only three pipeline stages for performing the computation tasks corresponding to the resolution level 1, level 2, and all the remaining levels, respectively. Two parameters, one specifying the design complexity from the point of view of synchronizing the operations of the stages and the other representing the utilization of the hardware resources of the pipeline, have been defined. It has been shown that the best combination for the value of these parameters is achieved when the pipeline is chosen to have three stages. In order to enhance the intra-stage parallelism, two main ideas have been employed for the internal design of each stage. The first idea is to divide the 2-D filtering operation into four subtasks that perform independently and simultaneously on the elements of even or odd indexed rows and columns of the 2-D input data. This idea stems from the fact that for each consecutive resolution level, the input data are decimated by a factor of two along the rows and columns of the 2-D data. Each subtask of the filtering operation is performed by a processing unit. The second idea employed is in organization of the array of bit-wise adders, which is the core of the

processing unit, in a way so as to minimize the delay of the critical path from a partial product input bit to a bit of an output sample through this array. In this chapter, this has been accomplished by minimizing the number of layers of the array while at the same time minimizing the delay of each layer.

In order to validate the proposed scheme, a circuit for the DWT computation has been designed, simulated and implemented in FPGA. The circuit is designed for a filter length $L=M=4$ and simulated for the number of the resolution levels $J=6$ and data size $N\times N=512\times512$. The simulation results have shown that the circuit designed based on the proposed scheme is able to operate at a maximum clock frequency $f_{max}=153$ MHz. The results of the FPGA implementation have shown that the circuit can process a 512×512 image in 0.97 ms, which is at least two times faster than that of the other FPGA implementations, and in some instances, even with less hardware utilization. Finally, it is worth noting that the architecture designed in this chapter is scalable in that its processing speed can be adjusted upward or downward by changing the number of MAC cells in each of the processing units by a factor equal to that of the reduction required in the processing speed.

Chapter 5

Conclusion

5.1 Concluding Remarks

The DWT is a computationally intensive transform because of the processing of large volumes of data at multiple resolution levels involved in its computation. Therefore, it is imperative to design efficient VLSI architectures to implement the DWT computation for real-time applications, especially those requiring processing of high-frequency signals or broadband data. Many pipeline architectures focusing on providing high computational speed or efficient hardware utilization have been proposed in the literature. However, these architectures have not exploited in their designs the features inherent in the definition of the DWT to the extent possible. Consequently, the speed provided by these architectures is not commensurate with the amount of hardware utilized by them.

The objective of this thesis has been to develop a scheme for the design of hardware resource-efficient high-speed pipeline architectures for the computation of the 1-D and 2-D DWT. The goal of high speed has been achieved by maximizing the operating frequency and minimizing the number of clock cycles required for the DWT computation,

which in turn, have been realized by enhancing the inter-stage and intra-stage parallelisms of pipeline architectures.

In order to enhance the inter-stage parallelism, a study has been undertaken for determining the number of pipeline stages required for the DWT computation so as to synchronize their operations while providing to each stage the amount of hardware resources proportional to the task assigned to it. This study has determined that employment of two pipeline stages with the first one performing the task of the first resolution level and the second one that of all the other resolution levels of the 1-D DWT computation, and employment of three pipeline stages with the first and second ones performing the tasks of the first and second resolution levels and the third one performing that of the remaining resolution levels of the 2-D DWT computation, are the optimum choices for the development of 1-D and 2-D pipeline architectures, respectively.

With the number of pipeline stages as determined above, coupled with the fact that the nature of the filtering operations required in all the subbands and resolution levels is the same, the intra-stage parallelism has been enhanced by employing the following two main ideas. The first idea, which stems from the fact that in each consecutive resolution level the input data are decimated by a factor of two along each of the data dimensions, is that the filtering operations of a stage can be conveniently divided into a certain number of subtasks (two subtasks for the 1-D data and four subtasks for the 2-D data) that can be performed in parallel by operating on even- and odd-numbered samples along each dimension of the samples. Each subtask is an FIR filtering operation performing a set of multiply-accumulate operations, which can be accomplished by employing a MAC-cell network consisting of a two-dimensional array of bit-wise adders. The second idea in

enhancing the intra-stage parallelism has been the design of this network so as to minimize its critical path. This has been achieved by maximally extending the bit-wise addition operations of the network horizontally through a suitable arrangement of half, full and specifically designed double adders.

In order to validate the proposed scheme for the design of pipeline architectures, two specific design examples have been considered, one for the 1-D DWT computation and the other for the 2-D DWT computation. For the 1-D case, a pipeline architecture has been designed to compute a 7-level DWT using 6-tap 1-D filters and simulated using 0.35-micron technology, whereas for the 2-D case, a pipeline architecture has been designed for the computation of a 6-level DWT using 4×4-tap 2-D filters and simulated using 0.18-micron technology. The simulation results for the 1-D example have shown that the architecture designed is able to operate at a maximum clock frequency of 138 MHz. Furthermore, in comparison to other 1-D architectures designed using comparable amount of hardware resources, it provides at least 30% reduction in the computation time, and has an area-time product that is at least 45% smaller. The simulation results for the 2-D example have shown that the architecture designed is capable of operating at a maximum clock frequency of 153 MHz, and in comparison to other 2-D architectures using similar amount of hardware resources, it is at least two times faster, and has an area-time product that is at least 33% smaller. Finally, the two pipeline architectures have been implemented on a Xilinx FPGA board to test their performance in a real circuit environment. The test results have been found to be in conformity with those obtained from the simulations.

In conclusion, this thesis has been concerned with the design of hardware resource-efficient high-speed pipeline architectures for discrete wavelet transforms. In order to meet this objective, a number of novel ideas and schemes that enhance the inter- and intra-stage parallelisms of the pipeline stages have been advanced. The effectiveness of these ideas and schemes has been validated through designs, simulations and implementations of specific cases of 1-D and 2-D DWT architectures.

5.2 Scope for Future Work

In this thesis, a scheme has been proposed for the design of high-speed hardware-resource-efficient pipeline architectures for the DWT computation. In order to achieve this goal, certain characteristics inherent in the 1-D and 2-D discrete wavelet transforms have been exploited so as to maximize the inter- and intra-stage parallelisms of the pipeline stages. One could investigate the possibility of optimizing the proposed architectures further, in terms of their operating speed or the amount of hardware resources employed. Also, the scheme for the design of 1-D and 2-D pipeline architectures could be extended to higher dimensions.

In certain applications of the DWT, only some specific types of filters need to be employed [153]–[155]. For example, in speech recognition systems, 9/7 or 5/3 filters are often employed. The relationships that exist between the coefficients of the filters in various subbands could then be utilized to further enhance the operating speed of the pipelines.

In applications such as video compression, medical imaging and geographic data analysis, it is 1-D or 2-D wavelet transforms that are employed. The use of 1-D or 2-D

wavelet architectures in such applications results in high latency. A study could be undertaken to extend the scheme proposed in this thesis for the design of 1-D and 2-D pipeline architectures to 3-D pipeline architectures with a view to increase the processing speed of the applications involving 3-D data.

References

[1] S. M. M. Rahman, M. O. Ahmad, and M. N. S. Swamy, "A new statistical detector for DWT-based additive image watermarking using the Gauss-Hermite expansion," *IEEE Trans. Image Processing*, vol. 18, no. 8, pp. 1782–1796, Aug. 2009.

[2] S. M. M. Rahman, M. O. Ahmad, and M. N. S. Swamy, "Bayesian wavelet-based image denoising using the Gauss–Hermite expansion," *IEEE Trans. Image Processing*, vol. 17, no. 10, pp. 1755–1771, Oct. 2008.

[3] S. M. M. Rahman, M. O. Ahmad, and M. N. S. Swamy, "Video denoising based on inter-frame statistical modeling of wavelet coefficients," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 17, no. 2: pp. 187–198, Feb. 2007.

[4] M. I. H. Bhuiyan, M. O. Ahmad, and M. N. S. Swamy, "Spatially-adaptive thresholding in wavelet domain for despeckling of ultrasound images," *IET Image Processing*, vol. 3, pp. 147–162, Jun. 2009.

[5] M. I. H. Bhuiyan, M. O. Ahmad, and M. N. S. Swamy, "Spatially adaptive wavelet-based method using the Cauchy prior for denoising the SAR Images," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 17, no. 4, pp. 500–507, Apr. 2007.

[6] N. Gupta, *Video modeling and noise reduction in wavelet domain*, Ph.D. Thesis, Concordia University, Montreal, Nov. 2011.

[7] N. Gupta, M. N. S. Swamy, and E. I. Plotkin, "Despeckling of medical ultrasound images using data and rate adaptive lossy compression," *IEEE Trans. Medical Imaging*, vol. 24, pp. 743–754, Jun. 2005.

[8] S. Yu, M. O. Ahmad, and M. N. S. Swamy, "Video denoising using motion compensated 3-D wavelet transform with integrated recursive temporal filtering," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 20, no. 6, pp.780–791, Jun. 2010.

[9] X. Zhang and W.-P. Zhu, "Wavelet domain image restoration using adaptively regularized constrained total least squares," in *Proc. IEEE International Symposium Intelligent Multimedia, Video and Speech Processing*, 20−22 Oct. 2004, pp. 567−570.

[10] N. Gupta, M. N. S. Swamy, and E. I. Plotkin, "Wavelet domain-based video noise reduction using temporal discrete cosine transform and hierarchically adapted thresholding," *IET Image Processing,* vol. 1, no. 1, pp. 2−12, Mar. 2007.

[11] N. Gupta, M. N. S. Swamy, and E. I. Plotkin, "Temporally-adaptive MAP estimation for video denoising in the wavelet domain," in *Proc. IEEE International Conference Image Processing,* 8−11 Oct. 2006, pp. 1449−1452.

[12] C. Wu, W. P. Zhu, and M. N. S. Swamy, "A watermark embedding scheme in wavelet transform domain," in *Proc. IEEE Region 10 Conf. (TENCON)*, Chiang Mai, Thailand, 21−24 Nov. 2004, vol. 1, pp. 279−282.

[13] L. Ghouti, A. Bouridane, M. K. Ibrahim, and S. Boussakta, "Digital image watermarking using balanced multiwavelets" , *IEEE Trans. Signal Processing*, vol. 54, no. 4, pp. 1519−1536, Apr. 2006.

[14] S. G. Chang, B. Yu, and M. Vattereli, "Adaptive wavelet thresholding for image denoising and compression," *IEEE Trans. Image Processing*, vol. 9, pp. 1532−1546, Sep. 2000.

[15] Y. Wang, J. F. Doherty, and R. E. Van Dyck, "A wavelet-based watermarking algorithm for ownership verification of digital images," *IEEE Trans. Image Processing*, vol. 11, no. 2, pp. 77−88, Feb. 2002.

[16] K. Sayood, *Introduction to Data Compression*, San Mateo, CA, Morgan Kaufman, 2000.

[17] D. Sinha and A. Tewfik, "Low bit rate transparent audio compression using adapted wavelets," *IEEE Trans. Signal Processing*, vol. 41, no. 12, pp. 3463−3479, Dec. 1993.

[18] M. F. Akorede and H. Hizam, "Wavelet transforms: practical applications in power Systems", *Journal of Electrical & Technology*, vol. 4, no. 2, pp. 168–174, 2009.

[19] M. Vetterli and J. Kovačević, *Wavelets and subband coding*, Prentice Hall-PTR Englewood Cliffs, New Jersey, 1995.

[20] C.S. Burrus, R.A. Gopinath, and H. Guo, *Introduction to Wavelets and Wavelet Transforms: A Primer*, Prentice Hall, New Jersey, 1998.

[21] Wavelet. Wikipedia, 2004. Retrieved on April 14, 2009 from http://en.wikipedia.org/wiki/wavelet

[22] M. Sifuzzaman1, M.R. Islam1, and M.Z. Ali, "Application of wavelet transform and its advantages compared to Fourier transform," *Journal of Physical Sciences*, vol. 13, pp. 121–134, 2009.

[23] R. Polikar, "*The Wavelet Tutorial*," 1994. Retrieved on April 14, 2008 from http://engineering.rowan.edu/~polikar/WAVELETS/Wttutorial.html

[24] M. Misiti, Y. Misiti, G. Oppenheim, and J. M. Poggi, "*Wavelet Toolbox for use with MATLAB®*," MathWorks, 2002.

[25] MATLAB, Mathworks, Inc. Natick, MA, U.S.A. 1996–1997. Retrieved on September 14, 2009 from http://www.mathworks.com

[26] S. Subbarayan and S. K. Ramanathan, "Effective watermarking of digital audio and image using Matlab technique," in *Proc. 2nd International Conference Machine Vision*, Dec. 2009, pp. 317–319.

[27] C. Kumar, C. Shekhar, A. Soni, and B. Thakral, "Implementation of audio signal by using wavelet transform," *International Journal of Engineering Science and Technology*, vol. 2(10), pp. 4972–4977, 2010.

[28] G. Bi, "On computation of the discrete W transform," *IEEE Trans. Signal Processing*, vol. 47 no. 5, pp. 1450–1453, May 1999.

[29] S. Rioul and P. Duhamel, “Fast algorithm for discrete and continuous wavelet transforms,” *IEEE Trans. Information Theory,* vol. 38, no. 2, pp. 569−586, Mar. 1992

[30] M. D. Adams and R. K. Ward, “Symmetric-extension-compatible reversible integer-to-integer wavelet transforms,” *IEEE Trans. Signal Processing,* vol. 51, no. 10, pp. 2624−2636, Oct. 2003.

[31] J. Oliver and M. Perez Malumbres, “On the design of fast wavelet transform algorithms with low memory requirements,” *IEEE Trans. Circuits and Systems for Video Technology,* vol. 18, no. 2, pp. 237−248, Feb. 2008.

[32] A. Shahbahrami, B. Juurlink, and S. Vassiliadis, “Implementing the 2-D wavelet transform on SIMD-enhanced general-purpose processors,” *IEEE Trans. Multimedia*, vol. 10, no. 1, pp. 43−51, Jan. 2008.

[33] D. Chaver, C. Tenllado, L. Piñuel, M. Prieto, and F. Tirado, “2-D wavelet transform enhancement on general-purpose microprocessors: memory hierarchy and SIMD parallelism exploitation,” in *Proc. Int. Conf. High Performance Computing (ICHPC)*, 2002, pp. 9–21.

[34] G. Dimitroulakos, M. D. Galanis, A. Milidonis, and C. E. Goutis, “A high-throughput, memory efficient architecture for computing the tile-based 2-D discrete wavelet transform for the JPEG2000,” *Integration, the VLSI Journal*, vol. 39, pp. 1–11, Sep. 2005.

[35] C. Chrysytis and A. Ortega, “Line-based, reduced memory, wavelet image compression,” *IEEE Trans. Circuits and Systems for Video Technology*, vol. 9, no. 3, pp. 378–389, Mar. 2000.

[36] S. Zhuang, J. Carlsson, W. Li, K. Palmkvist, and L. Wanhammar, “GALS based approach to the implementation of the DWT filter bank,” in *Proc. IEEE 7th Int. Conf. Signal Processing (ICSP)*, 31 Aug.–4 Sep. 2004, vol. 1, pp. 567−570.

[37] G. Lafruit, F. Catthoor, J. P. H. Cornelis, and H. J. De Man, “An efficient VLSI architecture for 2-D wavelet image coding with novel image scan,” *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 56−68, Mar. 1999.

[38] C. Qi and G. F. Wang, "A wavelet-based parallel implementation for image encoding," in *Proc. IEEE 8th Int. Conf. Signal Processing*, 16−20 Nov. 2006, vol. 2, pp. 102−105

[39] O. Benderli, Y. C. Tekmen, and N. Ismailoglu, "A real-time, low latency, FPGA implementation of the 2-D discrete wavelet transformation for streaming image applications," in *Proc. Euromicro Symposium Digital System Design*, 1−6 Sep. 2003, pp. 384 – 389.

[40] A. Shahbahrami, B. Juurlink, and S. Vassiliadis, "Performance comparison of SIMD implementations of the discrete wavelet transform," in *Proc. IEEE 16th Int. Conf. Application-Specific Systems, Architectures Processors (ASAP)*, 23−25 Jul. 2005, pp. 393−398.

[41] M. Week and M. Bayoumi, "Discrete wavelet transform: architectures, design and performance issues," *Journal of VLSI Signal Processing*, vol. 35, no. 2, pp. 155−178, 2003.

[42] Y. Guo, H. Zhang, X. Wang, and J.R. Cavallaro, "VLSI implementation of Mallat's fast discrete wavelet transform algorithm with reduced complexity," in *Proc. IEEE Conf. Global Telecommunications (GLOBECOM)* 25−29 Nov. 2001, vol. 1, pp. 320−324.

[43] A. Grzesczak, M.K. Mandal, and S. Panchanathan, "VLSI implementation of discrete wavelet transform," *IEEE Trans. Very Large Scale Integration Systems*, vol. 4, no. 4, pp. 421−433, Dec. 1996.

[44] H. Y. Liao, M. K. Mandal, and B. F. Cockburn, "Efficient architectures for 1-D and 2-D lifting-based wavelet transforms," *IEEE Trans. Signal Process.*, vol. 52, no. 5, pp. 1315–1326, May 2004.

[45] S. S. Nayak, "Bit-level systolic implementation of 1D and 2D discrete wavelet transform," *IEE Proceedings Circuits, Devices and Systems*, vol. 152, no. 1, pp. 25−32, Feb. 2005.

[46] T. C. Denk and K. K Parhi, "Systolic VLSI architectures for 1-D discrete wavelet transforms," in *Record of the Thirty-Second Asilomar Conference Signals, Systems and Computers*, 1–4 Nov. 1998, vol. 2, pp. 1220–1224.

[47] A.S. Lewis and G. Knowles, "VLSI architecture for 2D Daubechies wavelet transform without multipliers," *Electron. Lett.*, vol. 27, no. 2, pp.171–173, Jan. 1991.

[48] S. Movva, S. Srinivasan, "A novel architecture for lifting-based discrete wavelet transform for JPEG2000 standard suitable for VLSI implementation," in *Proc. IEEE 16th Int. Conf. VLSI Design*, 4–8 Jan. 2003, pp. 202- 207.

[49] K.C. Hung, Y.S. Hung, and Y.J. Huang, "A nonseparable VLSI architecture for the two-dimensional discrete periodized wavelet transform," *IEEE Trans. VLSI Systems*, vol. 9, no. 5, pp. 565–576, Oct. 2001.

[50] I. S. Uzun and A. Amira, "Design and FPGA implementation of non-separable 2-D biorthogonal wavelet transforms for image/video coding," in *Proc. IEEE Int. Conf. Image Processing (ICIP)*, 24–27 Oct. 2004, vol.4, pp. 2825–2828.

[51] P. K. Meher, B. K. Mohanty, and J. C. Patra, "Hardware-efficient systolic-like modular design for two-dimensional discrete wavelet transform," *IEEE Trans. Circuits Syst. II*: *Express Briefs,* vol. 55, no. 2, pp. 151–155, Feb. 2008.

[52] M. Vishwanath, "The recursive pyramid algorithm for the discrete wavelet transform," *IEEE Trans. Signal Processing*, vol. 42, no. 3, pp. 673–677, Mar. 1994.

[53] C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: from single chip implementations to mapping on SIMD array computers," *IEEE Trans. Signal Processing*, vol. 43, no. 3, pp. 759–771, Mar. 1995.

[54] K. K. Parhi and T. Nishitani, "VLSI architectures for discrete wavelet transforms," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 2, pp. 191–202, Jun. 1993.

[55] S. Masud and J.V. McCanny, "Reusable silicon IP cores for discrete wavelet transform applications," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 6, pp. 1114–1124, Jun. 2004.

[56] M. A. Farahani, M. Eshghi, "Architecture of a wavelet packet transform using parallel filters," in *Proc. IEEE Region 10 Conf.* (TENCON), 14–17 Nov. 2006, pp.1–4.

[57] C. Chakrabarti and C. Mumford, "Efficient realizations of analysis and synthesis filters based on the 2-D discrete wavelet transform," in *Proc. IEEE Int. Conf. Audio, Speech, and Signal Processing*, May 1996, pp. 3256–3259.

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

[59] Y. Chen, Z. L. Yang, T. C. Wang, and L. G. Chen, "A programmable parallel VLSI architecture for 2D discrete wavelet transform," *Journal of VLSI Signal Processing*, vol. 28, pp. 151–163, Jul. 2001.

[60] F. Marino, V. Piuri, and E. E. Swartzlander, "A parallel implementation of the 2-D discrete wavelet transform without interprocessor communications," *IEEE Trans. Signal Processing*, vol. 47, no. 11, pp. 3170–3184, Nov. 1999.

[61] F. Marino, D. Guevorkian, and J. Astola, "Highly efficient high-speed/low-power architectures for 1-D discrete wavelet transform," *IEEE Trans. Circuits Syst. II*: *Analog and Digital Signal Processing*, vol. 47, no. 12, pp. 1492–1502, Dec. 2000.

[62] T. Park, "Efficient VLSI architecture for one-dimensional discrete wavelet transform using a scalable data recorder unit," in *Proc. the International Technical Conf. Circuits and Systems, Computers and Communications (ITC CSCC)*, Phuket, Thailand, 16–19 Jul. 2002, pp.353–356.

[63] P. Y. Chen, "VLSI implementation for one-dimensional multilevel lifting-based wavelet transform," *IEEE Trans. Computers*, vol. 53, no. 4, pp. 386–398, Apr. 2004.

[64] J. M. Jou, P. Y. Chen, Y. H. Shiau, and M. S. Liang, "A scalable pipelined architecture for separable 2-D discrete wavelet transform," in *Proc. the Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 18−21 Jan. 1999, vol. 1, pp. 205−208.

[65] K. Mihić, "An efficient semi-systolic architecture for 2-D discrete wavelet transform," in *Proc. IEEE European Conf. Circuit Theory and Design (ECCTD)*, Espoo, Finland, 28−31 Aug. 2001, pp. 333−336.

[66] F. Marino, "Efficient high-speed/low-power pipelined architecture for the direct 2-D discrete wavelet transform," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 12, pp. 1476−1491, Dec. 2000.

[67] F. Marino, "Two fast architectures for the direct 2-D discrete wavelet transform," *IEEE Trans. Signal Processing*, vol. 49, no. 6, pp. 1284−1259, Jun. 2001.

[68] L. Debnath, *Wavelet transforms and their applications*, Springer, 2001.

[69] G. Strang and T. Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press: 1996.

[70] I. Daubechies, *Ten lectures on wavelets*, Philadelphia (PA), SIAM, 1992.

[71] S. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Trans. Pattern Analysis and Machine Intell.*, vol. 11, no. 7, pp. 674−693, Jul. 1989.

[72] C. Valens, A real friendly guide to wavelets, http://polyvalens.pagesperso-orange.fr/clemens/wavelets/wavelets.html

[73] A. Jense and A. La Cour-Harbo, *Ripples in Mathematics: the Discrete Wavelet Transform*, Springer, 2001.

[74] G. Knowles, "VLSI architecture for the discrete wavelet transform," *Electron. Lett.*, vol. 26, no. 15, pp.1184−1185, Jul. 1990.

[75] M. Vishwanath, R. Owens, and M. J. Irwin, "VLSI architectures for the discrete wavelet transform," *IEEE Trans. Circuits Syst. II*: *Analog and Digital Signal Processing,* vol. 42, no. *5,* pp. 305−316, May 1995.

[76] C. Chakrabarti, M. Vishwanath, and R. M. Owens, "Architectures for wavelet transforms: a survey," *Journal of VLSI Signal Processing*, vol. 14, no. 2, pp. 171−192, Feb. 1996.

[77] I. S. Uzun, A. Amira, and A. Bouridane, "An efficient architecture for 1-D discrete biorthogonal wavelet transform," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 23−26 May 2004, vol.2, pp. (II)697−700.

[78] T. C. Denk and K. K. Parhi, "Synthesis of folded pipelined architectures for multirate DSP algorithms," *IEEE Trans. Very Large Scale Integration Systems,* vol. 6, no. 4, pp. 595−607, Dec. 1998.

[79] T. Cooklev, "An efficient architecture for orthogonal wavelet transforms," *IEEE Signal Processing Letters*, vol. 13, no. 2, pp. 77−79, Feb. 2006.

[80] E. Huluta, E. M. Petriu, S. R. Das and A. H. Al-Dhaher, "Discrete wavelet transform architecture using fast processing elements," in *Proc. IEEE 19th Conf. Instrumentation and Measurement Technology (IMTC)*, 21−23 May 2002, vol. 2, pp. 1537−1542.

[81] T. Acharya and P. Y. Chen, "VLSI implementation of a DWT architecture," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, 31 May−3 Jun. 1998, vol. 2, pp. 272−275.

[82] M. Weeks, J. Limqueco, and M. Bayoumi, "On block architectures for discrete wavelet transform," in *Record of the Thirty-Second Asilomar Conf. Signals, Systems and Computers*, 1−4 Nov. 1998, vol. 2, pp. 1022−1026.

[83] S. B. Pan and R. H. Park, "Systolic array architectures for computation of the discrete wavelet transform," *Journal of Visual Communication and Image Representation*, vol. 14, no. 3, pp. 217–231, Sep. 2003.

[84] M. Nibouche, A. Bouridane, and O. Nibouche, "Rapid prototyping of biorthogonal discrete wavelet transforms on FPGAs," in *Proc. IEEE 8th Int. Conf. Electronics, Circuits and Systems (ICECS)*, 02–05 Sep. 2001, vol. 3, pp. 1399–1402.

[85] M. Bahoura and H. Ezzaidi, "Real-time implementation of discrete wavelet transform on FPGA," in *Proc. IEEE 10th Int. Conf. Signal Processing (ICSP),* 24−28 Oct. 2010, pp. 191−194.

[86] T. Y. Sung, H. C. Hsin, Y. S. Shieh, and C. W. Yu, "Low-power multiplierless 2-D DWT and IDWT architectures using 4-tap Daubechies filters," in *Proc. IEEE 7th Int. Conf. Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, Dec. 2006, pp. 185−190.

[87] M. Martina and G. Masera, "Multiplierless, folded 9/7-5/3 wavelet VLSI architecture," *IEEE Trans. Circuits Syst. II*: *Express Briefs*, vol. 54, no. 9, pp. 770−774, Sep. 2007.

[88] S. B. Pan and R. H. Park, "VLSI architectures of the 1-D and 2-D discrete wavelet transforms for JPEG 2000," *Signal Process*ing, vol. 82, no. 7, pp. 981−992 Jul. 2002.

[89] X. D. Xu and Y. Q. Zhou; "Efficient FPGA Implementation of 2-D DWT for 9/7 Float Wavelet Filter," *Int. Conf. Information Engineering and Computer Science (ICIECS)*, 19−20 Dec. 2009, pp.1−4.

[90] C. Yu and S. J. Chen, "Design of an efficient VLSI architecture for 2-D discrete wavelet transforms," *IEEE Trans. Consumer Electronics*, vol. 45, no. 1, pp. 135−140, Feb. 1999.

[91] C. T. Huang, P. C. Tseng, and L. G. Chen, "Memory analysis and architecture for two-dimensional discrete wavelet transform," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, 17−21 May 2004, vol. 5, pp. (V)13−16.

[92] X. Tian, J. Wei, and J. Tian; "Memory-efficient architecture for fast two-dimensional discrete wavelet transform," in *Proc. IEEE Int. Conf. Computational Intelligence and Software Engineering (CiSE)*, 10−12 Dec. 2010, pp. 1−3.

[93] S. K. Paek, H. K. Jeon, and L. S. Kim, "Semi-recursive VLSI architecture for two dimensional discrete wavelet transform," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, 31 May−3 Jun. 1998, vol. 5, pp. 469−472.

[94] J. Chen and M. A. Bayoumi, “A scalable systolic array architecture for the 2D discrete wavelet transform,” in *Proc. IEEE Workshop VLSI Signal Processing*, Osaka, Japan, 16−18 Oct. 1995, vol. 3, pp.303−312.

[95] H. Y. H. Chuang and L. Chen, “VLSI architecture for the fast 2-D discrete orthonormal wavelet transform,” *Journal of VLSI Signal Processing*, vol. 10, pp. 225−236, 1995.

[96] T. Park and S. Jung, “High speed lattice based VLSI architecture of 2D discrete wavelet transform for real-time video signal processing,” *IEEE Trans. Consumer Electronics*, vol. 48, no. 4, pp. 1026−1032, Nov. 2002.

[97] J. C. Limqueco and M. A. Bayoumi, “A VLSI architecture for separable 2-D discrete wavelet transform,” *Journal of VLSI Signal Processing*, vol. 18, pp. 125, 1998.

[98] T. Huang, P. C. Tseng, and L. G. Chen, “Generic RAM-based architectures for two-dimensional discrete wavelet transform with line-based method,” *IEEE Trans. Circuits and Systems for Video Technology*, vol. 15, no. 7, pp. 910–920, Jul. 2005.

[99] M. Ravasi, L. Tenze, and M. Mattavelli, “A scalable and programmable architecture for 2-D DWT decoding,” *IEEE Trans. Circuits and Systems for Video Technology*, vol. 12, no. 8, pp. 671−677, Aug. 2002.

[100] C. Y. Chen, Z. L. Yang, T. C. Wamg, and L. G. Chen, “A programmable VLSI architecture for 2-D discrete wavelet transform,” in *Proc. IEEE Int. Symp. Circuits and Systems* (*ISCAS*), Geneva, Switzerland, 28−31 May 2000, vol. 1, pp. 619−622.

[101] B. K. Mohanty and P. K. Meher, “Bit-serial systolic architecture for 2-D non-separable discrete wavelet transform,” in *Proc. IEEE Int. Conf. Intelligent and Advanced Systems* (*ICIAS*), Kuala Lumpur, Malaysia, Nov. 2007, pp.1355−1358.

[102] D. Sersic and M. Vrankic, “2-D nonseparable wavelet filter bank with adaptive filter parameters,” in *Proc. IEEE Europen Signal Processing Conf. (EUSIPCO)*, 3−6 Sep. 2002, vol. 1, pp. 137−140.

[103] M. H. Sheu, M. Der Shieh, and S. W. Liu, "A low-cost VLSI architecture design for non-separable 2-D discrete wavelet transform," in *Proc. IEEE 40th Midwest Symp. Circuits and Systems*, 3–6 Aug. 1997, vol. 2, pp. 1217–1220.

[104] J. Kovacevic and M. Vetterli, "Nonseparable two- and three-dimensional wavelets," *IEEE Trans. Signal Processing*, vol. 43, no. 5, pp. 1269–1273, May 1995.

[105] C. Yu and S. J. Chen, "VLSI implementation of 2-D discrete wavelet trnasform for real-time video signal processing," *IEEE Trans. Consumer Electronics*, vol. 43, no. 4, pp. 1270–1279, Nov. 1997.

[106] B. Das and S. Banerjee, "VLSI architecture for a new real-time 3D wavelet transform," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, (ICASSP)*, Orlando, U.S.A., 13–17 May 2002, vol. 3, pp. (III)3224–3227.

[107] M. Weeks and M. A. Bayoumi, "Three-dimensional discrete wavelet transform architectures," *IEEE Trans. Signal Processing*, vol. 50, no. 8, pp. 2050–2063, Aug. 2002.

[108] B. K. Mohanty and P. K. Meher, "Parallel and pipeline architectures for high-throughput computation of multilevel 3-D DWT," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 20, no. 9, pp. 1200–1209, Sep. 2010.

[109] Q. Dai, X. Chen, and C. Lin, "A novel VLSI architecture for multidimensional discrete wavelet transform," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 14, no. 8, pp. 1105–1110, Aug. 2004.

[110] C. Huang, P. Tseng, and L. Chen, "Analysis and VLSI architecture for 1-D and 2-D discrete wavelet transform," *IEEE Trans. Signal Processing,* vol. 53, no. 4, pp. 1575–1586, Apr. 2005.

[111] D. Zervas, G. P. Anagnostopoulos, V. Spiliotopoulos, Y. Andreopoulos, and C. E. Goutis, "Evaluation of design alternatives for the 2-D-discrete wavelet transform," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 12, pp. 1246–1262, Dec. 2001.

[112] C. Tenllado, J. Setoain, M. Prieto, L. Piñuel, and F. Tirado, "Parallel implementation of the 2D discrete wavelet transform on graphics processing units: filter-bank versus lifting," *IEEE Trans. Parallel and Distributed Systems*, vol. 19, no. 3, pp. 299−310, Mar. 2008.

[113] K. A. Kotteri, S. Barua, A. E. Bell, and J. E. Carletta, "A comparison of hardware implementations of the biorthogonal 9/7 DWT: convolution versus lifting," *IEEE Trans. Circuits Syst. II*: *Express Briefs*, vol. 52, no. 5, pp. 256−260, May 2006.

[114] C. T. Huang, P. C. Tseng, and L. G. Chen, "Flipping structure: an efficient VLSI architecture for lifting-based discrete wavelet transform," *IEEE Trans. Signal Processing*, vol. 52, no. 4, pp. 1080−1089, Apr. 2004.

[115] T. C. Denk and K. K. Parhi, "VLSI architectures for lattice structure based orthonormal discrete wavelet transforms," *IEEE Trans. Circuits and Syst. II: Analog and Digital Signal Processing*, vol. 44, no. 2, pp. 129−132, Feb 1997.

[116] J. T. Kim, Y. H. Lee, T. Isshiki, and H. Kunieda, "Scalable VLSI architectures for lattice structure-based discrete wavelet transform," *IEEE Trans. Circuits and Syst. II: Analog and Digital Signal Process.*, vol. 45, no. 8, pp. 1031−1043, Aug. 1998.

[117] C. Xiong, J. Tian, and J. Liu; "Efficient architectures for two-dimensional discrete wavelet transform using lifting scheme," *IEEE Trans. Image Processing*, vol. 16, no. 3, pp. 607−614, Mar. 2007.

[118] C. Xiong, J. Tian, and J. Liu, "A fast VLSI architecture for two-dimensional discrete wavelet transform based on lifting scheme," in *Proc. IEEE 7th Int. Conf. Solid-State Integrated-Circuit Technology*, 1−4 Nov. 2004, vol. 2, pp. 1661.

[119] M. Ferretti and D. Rizzo, "A parallel architecture for the 2-D discrete wavelet transform with integer lifting scheme," *Journal of VLSI Signal Processing*, vol. 28, no. 3, pp.165−185, Jul. 2001.

[120] X. Lan, N. Zheng, and Y. Liu, "Low-power and high-speed VLSI architecture for lifting-based forward and inverse wavelet transform," *IEEE Trans. Consumer Electronics*, vol. 51, no. 2, pp. 379−385, May 2005.

[121] W. Jiang and A. Ortega, "Lifting factorization-based discrete wavelet transform architecture design," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 5, pp. 651−657, May 2001.

[122] C. Wang and W.S. Gan, "Efficient VLSI architecture for lifting-based discrete wavelet packet transform," *IEEE Trans. Circuits Syst. II*: *Express Briefs*, vol. 54, no. 5, pp. 422−426, May 2007.

[123] G. Shi, W. Liu, L. Zhang, and F. Li, "An efficient folded architecture for lifting-based discrete wavelet transform," *IEEE Trans. Circuits Syst. II*: *Express Briefs*, vol. 56, no. 4, pp. 290−294, Apr. 2009.

[124] M. Alam, W. Badawy, V. Dimitrov, and G. Jullien, "An efficient architecture for a lifted 2D biorthogonal DWT", *Journal of VLSI Signal Processing*, vol. 40, no. 3, pp. 333−342, Jul. 2005.

[125] J. Chilo and T. Lindblad, "Hardware implementation of 1D wavelet transform on an FPGA for infrasound signal classification," *IEEE Trans. Nuclear Science*, vol. 55, no. 1, pp. 9−13, Feb. 2008.

[126] S. Cheng, C. Tseng, and M. Cole, "Efficient and effective VLSI architecture for a wavelet-based broadband sonar signal detection system," in *Proc. IEEE 14th Int. Conf. Electronics, Circuits and Systems (ICECS)*, Marrakech, Morocco, Dec. 2007, pp. 593−596.

[127] K.G. Oweiss, A. Mason, Y. Suhail, A.M. Kamboh, and K.E. Thomson, "A scalable wavelet transform VLSI architecture for real-time signal processing in high-density intra-cortical implants ," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 6, pp. 1266−1278, Jun. 2007.

[128] A. Acharyya, K. Maharatna, B. M. Al-Hashimi, and S.R. Gunn, "Memory reduction methodology for distributed-arithmetic-based DWT/IDWT exploiting data symmetry," *IEEE Trans. Circuits Syst. II*: *Express Briefs*, vol. 56, no. 4, pp. 285−289, Apr. 2009.

[129] W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions," in *Proc. SPIE,* vol. 2569, pp.68−79, 1995.

[130] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *Journal of Fourier Analysis and Applications*, vol. 4, no. 3, pp. 247–269, 1998.

[131] C. Zhang, C. Wang, and M. O. Ahmad, "An efficient buffer-based architecture for on-line computation of 1-D discrete wavelet transform," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, Montreal, Canada, May 2004, vol. 5, pp. 201–204.

[132] C. Zhang, C. Wang, and M.O. Ahmad, "A VLSI architecture for a high-speed computation of the 1D discrete wavelet transform," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, May 2005, vol. 2, pp. 1461–1464.

[133] C. Zhang, C. Wang, and M. O. Ahmad, "A pipeline VLSI architecture for high-speed computation of the 1-D discrete wavelet transform," *IEEE Trans. Circuits Syst. I*, vol. 57, no. 10, pp. 2729–2740, Oct. 2010.

[134] M. Ferretti and D. Rizzo, "Handling borders in systolic architectures for the 1-D discrete wavelet transform for perfect reconstruction," *IEEE Trans. Signal Processing*, vol. 48, no. 5, pp. 1365–1378, May 2000.

[135] A. Satoh, N. Ooba, K. Takano, and E. D′Avignon, "High-speed MARS hardware," in *Proc. 3rd AES conf.*, New York, U.S.A., Apr. 2000, pp. 305–316.

[136] S. Masud and J.V. McCanny, "Rapid design of diorthogonal wavelet transforms," *IEE Proceedings Circuits, Devices and Systems*, vol. 147, no. 5, pp. 293–296, Oct. 2000.

[137] M. Nibouche, A. Bouridane, F. Murtagh, and O. Nibouche, "FPGA-based discrete wavelet transforms system," in *Proc. the 11th International Conference Field-Programmable Logic and Applications (FPL)*, London, UK, 2001, pp. 607–612.

[138] A. M. Al-Haj, "An FPGA-based parallel distributed arithmetic implementation of the 1-D discrete wavelet transform," *Journal of Informatica*, vol. 29, no. 2, pp. 241–247, 2005.

[139] C. Jing and H.-Y. Bin, "Efficient wavelet transform on FPGA using advanced distributed arithmetic," in *Proc. the 8th International Conference Electronic Measurement and Instruments (ICEMI)* 16 Aug.–18 Jul. 2007, vol. 2, pp. 512–515.

[140] V. Herrero, J. Cerdà, R. Gadea, M. Martínez, and A. Sebastià, "Implementation of 1-D Daubechies wavelet transform on FPGA," in *Proc. Online Symposium for Electronics Engineers (OSEE)*, Sep. 2007.

[141] Z.-G. Wu and W. Wang, "Pipelined architecture for FPGA implementation of lifting-based DWT," in *Proc. International Conference Electric Information and Control Engineering (ICEICE)*, 15–17 Apr. 2011, pp.1535–1538.

[142] C. Zhang, C. Wang, and M. O. Ahmad, "A VLSI architecture for a fast computation of the 2-D discrete wavelet transform," in *Proc. IEEE Int. Symp. Circuits and Systems* (*ISCAS*), May 2007, pp. 3980–3983.

[143] C. Zhang, C. Wang, and M. O. Ahmad, "A pipeline VLSI architecture for fast computation of the 2-D discrete wavelet transform," *IEEE Trans. Circuits Syst. I*, Digital Object Identifier: 10.1109/TCSI.2011.2180432, 2012.

[144] D. Guevorkian, P. Liuha, A. Launiainen, and V. Lappalainen, "Architectures for discrete wavelet transforms", U.S. Patent 6976046, December 13, 2005.

[145] J. Song and I. Park, "Pipelined discrete wavelet transform architecture scanning dual lines," *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 56, no. 12, pp. 916–920, Dec. 2009.

[146] C. Cheng and K. K. Parhi, "High-speed VLSI implementation of 2-D discrete wavelet transform," *IEEE Trans. Signal Processing*, vol. 56, no. 1, pp. 393–403, Jan. 2008.

[147] P. McCanny, S. Masud, and J. McCanny, "Design and implementation of the symmetrically extended 2-D wavelet transform," in *Proc. IEEE Int. Conf. Acoustic, Speech, and Signal Processing* (*ICASSP*), 13–17 May 2002, vol. 3, pp. 3108–3111.

[148] R. J. Palero, R. G. Gironez, and A. S. Cortes, “A novel FPGA architecture of a 2-D wavelet transform”, *Journal of VLSI Signal Processing*, vol. 42, no. 3, pp. 273−284, Mar. 2006.

[149] A. Benkrid, D. Crookes, and K. Benkrid, “Design and implementation of a generic 2-D biorthogonal discrete wavelet transform on an FPGA,” in *Proc. 9th Annual IEEE Symp. Field-Programmable Custom Computing Machines (FCCM)*, 29 Mar.–2 Apr. 2001, pp. 190−198.

[150] S. Raghunath and S. M. Aziz, “High speed area efficient multi-resolution 2-D 9/7 filter DWT processor,” in *Proc. IEEE Int. Conf. Very Large Scale Integration* (*IFIP*), Oct. 2006, vol. 16−18, pp. 210–215.

[151] M. Angelopoulou, K. Masselos, P. Cheung, and Y. Andreopoulos, “A comparison of 2-D discrete wavelet transform computation schedules on FPGAs,” in *Proc. IEEE Int. Conf. Field Programmable Technology* (*FPT*), Bangkok, Thailand, Dec. 2006, pp. 181–188.

[152] I. S. Uzun and A. Amira, “Rapid prototyping -- framework for FPGA-based discrete biorthogonal wavelet transforms implementation,” *IEE Proceedings Vision, Image and Signal Processing*, vol. 153, no. 6, pp. 721–734, Dec. 2006.

[153] B. F. Wu and Y. Q. Hu, “An efficient VLSI implementation of the discrete wavelet transform using embedded instruction codes for symmetric filters,” *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no. 9, pp. 936−943, Sep. 2003.

[154] G. Dillen, B. Georis, J. D. Legat, and O. Cantineau, “Combined line-based architecture for the 5-3 and 9-7 wavelet transform of JPEG2000,” *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no. 9, pp. 944−950, Sep. 2003.

[155] B. Das and S. Banerjee, “Data-folded architecture for running 3D DWT using 4-tap Daubechies filters,” *IEE Proceedings Circuits, Devices and Systems*, vol. 152, no. 1, pp. 17−24, 4 Feb. 2005.