



Efficient Architecture for Discrete Wavelet Transform Using Daubechies

Zhang Xiaoyin


**A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering
Prince of Songkla University**

2010

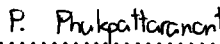
Copyright of Prince of Songkla University


Thesis Title Efficient Architecture for Discrete Wavelet Transform Using
Daubechies
Author Mr. Zhang Xiaoyin
Major Program Electrical Engineering


Major Advisor:



.....
(Asst. Prof. Dr. Nattha Jindapetch)

Examining Committee:

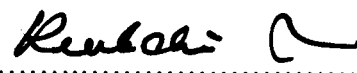

.....Chairperson
(Asst. Prof. Dr. Pornchai Phukpattaranont)


.....
(Asst. Prof. Dr. Nattha Jindapetch)


.....
(Asst. Prof. Dr. Wannarat Suntiamorntut)


.....
(Assoc. Prof. Dr. Wattanapong Kurdthongmee)

The Graduate School, Prince of Songkla University, has approved this thesis as partial fulfillment of the requirements for the Master of Engineering Degree in Electrical Engineering.


.....
(Assoc. Prof. Dr. Kerkchai Thongnoo)
Dean of Graduate School

Thesis Title Efficient Architecture for Discrete Wavelet Transform Using
Daubechies
Author Mr. Zhang Xiaoyin
Major Program Electrical Engineering
Academic Year 2010

ABSTRACT

A new basic architecture for one-dimensional discrete wavelet transform (1-D DWT) using Daubechies biorthogonal wavelet architecture is presented in this paper. The proposed DWT is composed of two independent FIR filters: a high-pass transposed form FIR filter and a low-pass transposed form FIR filter. The input to each filter is the same style as in the lifting scheme. Each FIR filter is a two-stage pipeline of which the fastest clock cycle is only either two adders delay or one multiplier delay. The proposed architecture has higher speed improved from Daubechies architecture, but uses less hardware resources. The area is further optimized by the RAG (Reduce Adder Graph) algorithm. Compare to the lifting scheme architecture which is commonly used, the proposed architecture achieves faster speed, shorter output latency, and the efficient pipeline architecture with simpler control.

Keywords: Discrete Wavelet Transform (DWT), Daubechies, Lifting scheme, digital filter, VLSI design.

ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to Asst. Prof. Dr. Nattha Jindapetch for her encouragement, teach and help.

Also thanks all the teachers in Faculty of Engineering who once offered me valuable courses and advice during my study. Specially thanks external committees Asst. Prof. Dr. Pornchai Phukpattaranont and Asst. Prof. Dr. Wannarat Suntiamorntut, and external committee Assoc. Prof. Dr. Wattanapong Kurdthongmee for coming to be a committee in my graduation thesis presentation.

Then, I feel grateful to my classmates in Department of Electrical Engineering for their kindness and help.

Zhang Xiaoyin

CONTENTS

Chapter 1	Introduction.....	1
1.1	Motivation.....	1
1.2	Research Background	1
1.2.1	Overview of Field Programmable Gate Arrays (FPGA)	1
1.2.2	Overview Dircrete Wavelet Transform (DWT)	1
1.3	Thesis Objective.....	4
1.4	Research Plan.....	6
Chapter 2	VLSI Architecture for Discrete Wavelet Transform	7
2.1	Introduction.....	7
2.2	Basic Daubechies Architecture	7
2.3	Lifting Implementation of the Discrete Wavelet Transform	9
2.4	Existing DWT Implementation.....	12
2.4.1	DWT VLSI Design by using Daubechies.....	12
2.4.2	DWT VLSI Design by using Lifting Scheme.....	15
2.5	RAG for High-Performance.....	24
2.6	Lifting 2-D DWT recursive Architecture [28].....	26
Chapter 3	The Proposed 1-D DWT Architecture	31
3.1	Expressions Formalization.....	31
3.2	1-D Architecture for High/Low pass Filter.....	32
3.3	Generalization for Even- and Odd-tap Filters.....	34
3.4	High-Speed Performance with RAG	36
3.5	Proposed ZXY 2-D DWT Architecture	37
Chapter 4	Performance Analysis and Comparison.....	38
Chapter 5	Conclusions.....	41

LIST OF TABLES

Table 3.1	The dataflow for the ZXY DWT 7-taps high-pass filter, about Fig. 3.1 ..34
Table 3.2	The used resources for even and odd number taps filter35
Table 4.1	Performance Comparison for 1-D DWT 9/7 and 5/3-tap filters.....38
Table 4.2	Performance Comparison for DWT 9/7 tap filter by Xilinx ISE 11.....40

LIST OF FIGURES IN TEXT

Figure 1.1	First level of decomposition [12]	2
Figure 1.2	One octave (Third level) of 2D-DWT	2
Figure 1.3	Basic DWT by 9/7-tap Daubechies FIR filter [2]	3
Figure 1.4	Lifting DWT	4
Figure 1.5	ZXY 1D-DWT by 9/7 taps FIR Filter	5
Figure 2.1	Basic DWT architecture by 9/7-tap Daubechies FIR filter	8
Figure 2.2	Lifting based forward and inverse DWT.....	10
Figure 2.3	Fast 9/7 direct implementation[14]	13
Figure 2.4	Fast modified 9/7 direct implementation[14]	13
Figure 2.5	DWT filter bank of Transposed form FIR [15].....	14
Figure 2.6	The horizontal low-pass filter for 1-D DWT [16].....	15
Figure 2.7	The shift and adder circuits replacing the multiplier [16].....	15
Figure 2.8	The direct mapped architecture [17].....	17
Figure 2.9	The folded architecture in [18].....	18
Figure 2.10	Data-flow and register allocation of the MAC based architecture in [20]	19
Figure 2.11	A flipping architecture [21]. (a) Original architecture, (b)-(c) Scaling the coefficients to reduce the number of multiplications, (d) Splitting the three-input addition nodes to two-input nodes	20
Figure 2.12	Processor assignment and partial schedule for the (5,3) filter implementation in the Generalized architecture in [22]	21
Figure 2.13	Processor architecture for the (5,3) and (9,7) filters in [22].....	22
Figure 2.14	The recursive architecture in [24].....	23
Figure 2.15	Realization of F6 using RAG algorithm [27].....	25
Figure 2.16	Calculation sequence of the 2-D RA.....	26
Figure 2.17	System architecture [28].....	27
Figure 2.18	A one-level 2-D DWT [28]	27
Figure 2.19	Implementation of a Lifting step. [28]	28
Figure 2.20	Architecture of the row processor [28].....	29
Figure 2.21	Architecture of the column processor [28].....	29

Figure 3.1	The proposed 1-D DWT architecture for 7-tap high-pass filter	33
Figure 3.2	The proposed 1-D DWT with 4-tap (even number) filter	35
Figure 3.3	Proposed 1-D DWT by 9/7 taps filter and its multiplier block design....	36
Figure 3.4	ZXY parallel architecture for column processing (7-tap)	37
Figure 4.1	Simulation for Daubechies DWT by 9/7 tap filter	39
Figure 4.2	Simulation for Lifting DWT by 9/7 tap filter	40
Figure 4.3	Simulation for ZXY DWT by 9/7 tap filter.....	40

Chapter 1. Introduction

1.1. Motivation

DWT (Discrete Wavelet Transform) is a popular mathematical method for many applications. In VLSI there are two different circuit structures for DWT-based designing, the conventional one is Daubechies-based architecture, and the other one is Lifting-based architecture which was improved from Daubechies. The Lifting-based is better than Daubechies on high-speed and reduction resource, but complicated control. This motivated us to design a new structure not only good at speed and resource but also easy to control.

1.2. Research Background

1.2.1 Overview of Field Programmable Gate Arrays (FPGA)

As Programmable Logic Devices (PLDs), FPGA is a programmable hardware that can easily implement designs with million of gates on a single chip. Designing time and cost can be substantially decreased when compares to the equivalent custom VLSI chips by the easy programmable characterize of the FPGAs, so that reconfigurable system is capable to be developed for executing application at performance.

1.2.2 Overview Discrete Wavelet Transform (DWT)

The wavelet transform is a linear transform that can operate in direct or inverse form. The wavelet transform approximates a function by representing it as a

linear combination of two sets of coefficients g and h constructed from functions derived respectively by a scaling function $\phi(t)$ and a mother wavelet function $\psi(t)$ [1].

The two-dimensional wavelet transform is computed by recursive application of one-dimensional wavelet transform. In a 1D-DWT, each octave computes two sub-bands from one original band and each of this sub-bands has a half number of coefficients input without data loss. In a 2D-DWT, each octave computes four sub-bands and each of these ones has a quarter number of coefficient input. Fig. 1.1 and Fig. 1.2 show the input and the output of one octave of 2D-DWT.

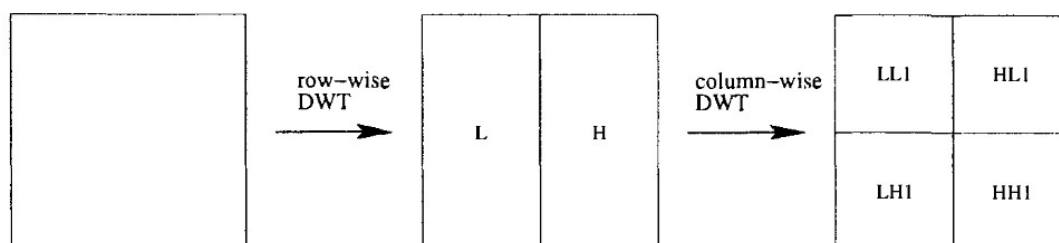


Fig. 1.1 First level of decomposition [12].

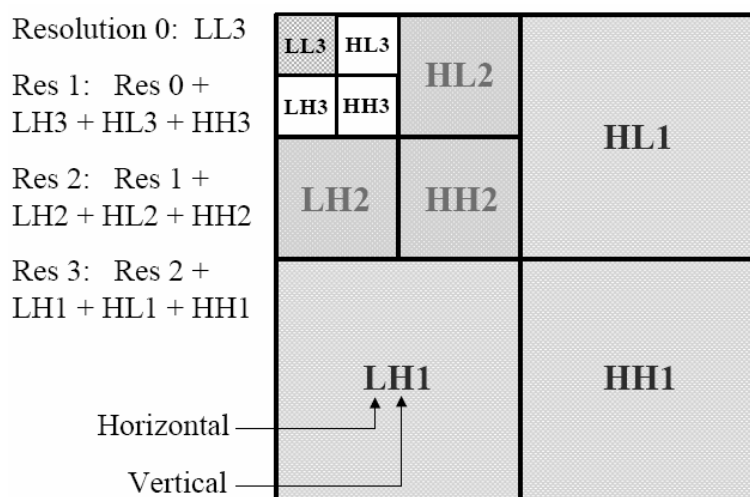


Fig. 1.2 One octave (Third level) of 2D-DWT.

The output coefficients from 1D-DWT are produced by application of two filters on data input samples, then producing two different output coefficient bands. A low pass filter using $h(x)$ coefficients produces an output band representing

low-frequency data and a high-pass filter using $g(x)$ coefficients produces an output band representing high frequency data [2].

The finite samples filtering presents a problem of discontinuities at its boundaries. Thus, the image boundary information could be lost if it was not treated properly. A simple method to eliminate this problem is mirroring the boundaries of the samples. The amount of samples mirroring depends on the depth of the low pass filter. The implementation of an irreversible DWT can be done by using the Daubechies biorthogonal wavelet coefficients, consisting of FIR filter coefficients for a 7-tap high-pass filter and a 9-tap low-pass filter, as shown in Fig. 1.3 [2].

The basic architecture defined in Fig. 1.3 requires 14 adders, 16 multipliers and 9 registers. This architecture has high area cost for a parallel implementation, thus several algorithms were developed to reduce this area cost [2].

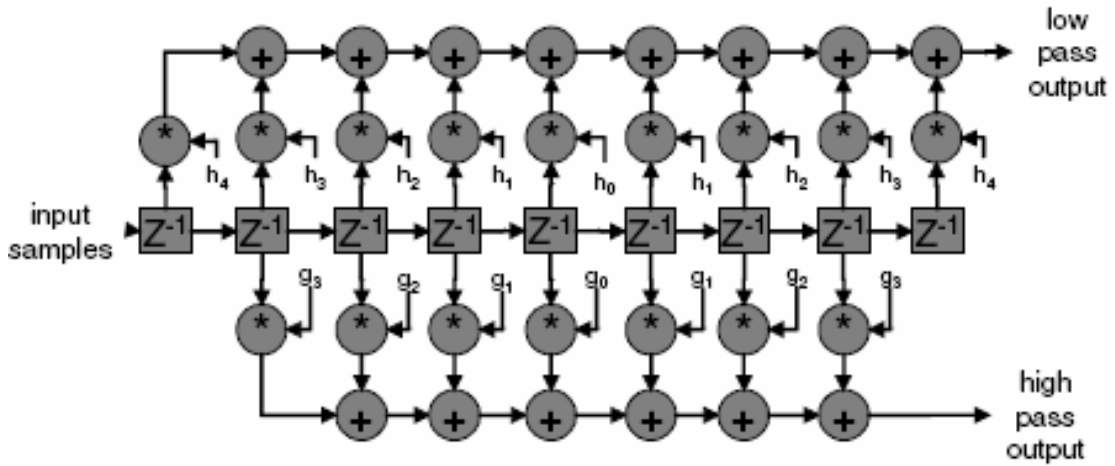


Fig. 1.3 Basic DWT by 9/7-tap Daubechies FIR filter [2].

Equation for 9-tap Low-pass:

$$\tilde{A}_{(n-2)} = h_4 Z_{2n-8} + h_3 Z_{2n-7} + h_2 Z_{2n-6} + h_1 Z_{2n-5} + h_0 Z_{2n-4} + h_1 Z_{2n-3} + h_2 Z_{2n-2} + h_3 Z_{2n-1} + h_4 Z_{2n} \quad (1.1)$$

Equation for 7-tap High-pass:

$$\tilde{C}_{(n-2)} = g_3 Z_{2n-6} + g_2 Z_{2n-5} + g_1 Z_{2n-4} + g_0 Z_{2n-3} + g_1 Z_{2n-2} + g_2 Z_{2n-1} + g_3 Z_{2n} \quad (1.2)$$

The lifting DWT scheme presented in [3] has reduced computational complexity, so reducing the hardware cost to compute the DWT. This algorithm

shown schematically in Fig. 1.4 was developed by a factorization of poliphase matrix from 9/7 Daubechies wavelet filter coefficients.

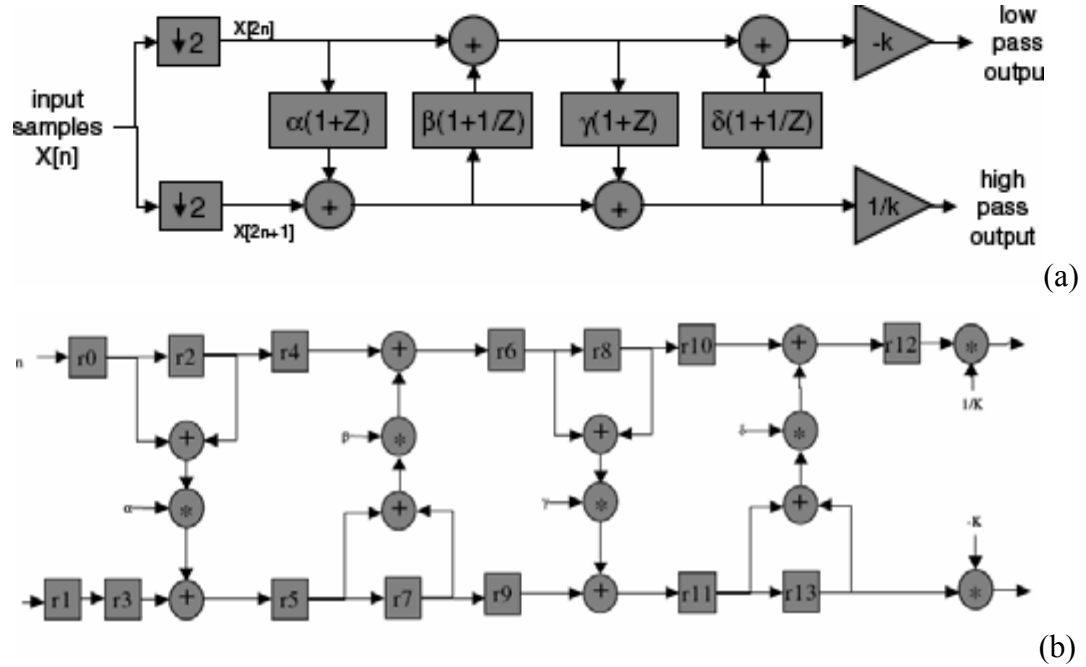


Fig. 1.4 Lifting DWT.

1.3. Research Methodology

In this thesis, we propose an efficient pipeline architecture based on the Daubechies architecture. The proposed DWT is composed of two independent FIR filters: a high-pass FIR filter and a low-pass FIR filter. Each FIR filter is a two-stage pipeline with the fastest clock cycle of only either two adders delay or one multiplier delay. The propose DWT does not use the extra shift registers for storing the previous sampled data. It uses only two registers for the even input and the odd input. In additions, it also does not need extra shift registers for pipeline stages. Therefore, the proposed architecture is as simple as the Daubechies architecture, but uses less hardware resources. The architecture can achieve the higher speed than the lifting architecture because the clock cycle is one multiplier delay or two adders delay, not one multiplier delay plus two adders delay in the lifting architecture. The proposed architecture still has larger area for the implementation using fixed-point adders and multipliers, but occupies less area than Lifting architecture for the implementation by

using the reduce adder graph (RAG) algorithm [4]. The proposed architecture can be further classified according to two structure types: even-tap filters and odd-tap filters. The multipliers occupation of even-tap filter is around 40% more than odd-tap filter.

My design is called ZXY improved from the Daubechies FIR Filter. This design used 9 multipliers, 15 adders and around 9 registers while the clock speed is the same as the Lifting scheme, or uses 18 registers for double speed as Fig. 1.5. Actually, this one can do more improving on the speed and area by RAG algorithm [4].

For example, the developed 9/7 architecture is shown in Fig. 1.5. where the equation are the same as Fig. 1.4's.

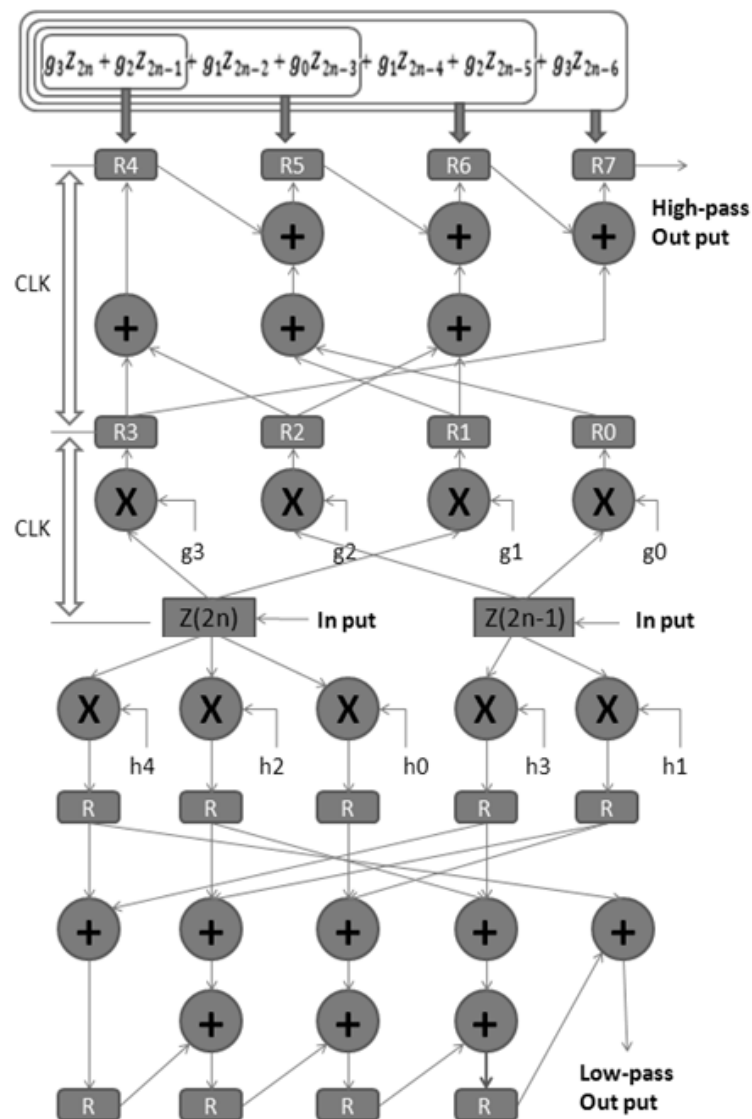


Fig. 1.5 ZXY 1D-DWT by 9/7 taps FIR Filter.

1.4. Research Plan

Task	Time(month)									
	1	3	4	5	6	9	11	14	17	
	2				7	10	12	15	18	
					8		13	16		
Study Verilog Hardware Description Language	■									
Study a commercial software(Xilinx ISE)	■	■								
Study JPEG2000 and DWT			■	■	■					
Develop algorithm for DWT					■	■				
Develop algorithm for hardware into a structural					■					
Test and improve the software						■	■	■		
Analyze and Conclusion							■	■	■	
Report									■	■

Note: The first month August 21, 2008

Chapter 2. VLSI Architectures for Discrete Wavelet Transforms

2.1. Introduction

The discrete wavelet transform (DWT) is widely used in many applications, especially in image compression algorithms such as the JPEG2000 standard [5] [6] [7] [8]. The DWT can be designed by any sub-band multiresolution. The implementations of one-dimensional discrete wavelet transform (1-D DWT) by filter banks have been presented in [9] [10]. The more conventional approach implementation is an irreversible DWT design by using lifting scheme coefficients [11] rather than the original Daubechies biorthogonal wavelet coefficients [1]. Based on lifting scheme factorization operation [3] for DWT, it requires much less multipliers, adders and storage elements compared to the Daubechies which uses convolution-based algorithm. However, the Daubechies architecture provides a simpler approach for DWT [12].

2.2. Basic Daubechies Architecture

A basic implementation of a 1-D DWT has been done by using the Daubechies biorthogonal wavelet coefficients [12]. Two different output bands are produced by applying two FIR filters on data input samples. A low-pass filter using $h(x)$ coefficients produces low-frequency data and a high-pass filter using $g(x)$ coefficients produces high-frequency data. As an example, Fig. 2.1 shows the 9/7-tap Daubechies DWT consisting of a 7-tap high-pass filter and a 9-tap low-pass filter.

The discrete signal input is $\mathbf{x} = \{x_1, x_2, x_3, \dots, x_n\}$. \mathbf{h}_i and \mathbf{g}_i are the low-pass and the high-pass filter coefficients, respectively. For example 9/7-tap filter has 9 low-pass filter coefficients $\mathbf{h}_i = \{h_{-4}, h_{-3}, h_{-2}, h_{-1}, h_0, h_1, h_2, h_3, h_4\}$ and 7

high-pass filter coefficients $\mathbf{g}_1 = \{g_{-3}, g_{-2}, g_{-1}, g_0, g_1, g_2, g_3\}$. Since the low-pass and the high-pass filter coefficients are symmetric ($h_i = h_{-i}, g_i = g_{-i}$), so, the output samples of the low-pass sub-band are as follows.

$$\begin{aligned} \mathbf{a}_1 &= h_0x_1 + h_1x_2 + h_2x_3 + h_3x_4 + h_4x_5 \\ \mathbf{a}_2 &= h_2x_1 + h_1x_2 + h_0x_3 + h_1x_4 + h_2x_5 + h_3x_6 + h_4x_7 \\ \mathbf{a}_3 &= h_4x_1 + h_3x_2 + h_2x_3 + h_1x_4 + h_0x_5 + h_1x_6 + h_2x_7 + h_3x_8 + h_4x_9 \\ \mathbf{a}_4 &= h_4x_3 + h_3x_4 + h_2x_5 + h_1x_6 + h_0x_7 + h_1x_8 + h_2x_9 + h_3x_{10} + h_4x_{11} \\ &\vdots \end{aligned} \tag{2.1}$$

Similarly, the high-pass sub-band equations can be rearranged as follows.

$$\begin{aligned} \mathbf{c}_1 &= g_1x_1 + g_2x_2 + g_3x_3 \\ \mathbf{c}_2 &= g_1x_1 + g_0x_2 + g_1x_3 + g_2x_4 + g_3x_5 \\ \mathbf{c}_3 &= g_3x_1 + g_2x_2 + g_1x_3 + g_0x_4 + g_1x_5 + g_2x_6 + g_3x_7 \\ \mathbf{c}_4 &= g_3x_3 + g_2x_4 + g_1x_5 + g_0x_6 + g_1x_7 + g_2x_8 + g_3x_9 \\ &\vdots \end{aligned} \tag{2.2}$$

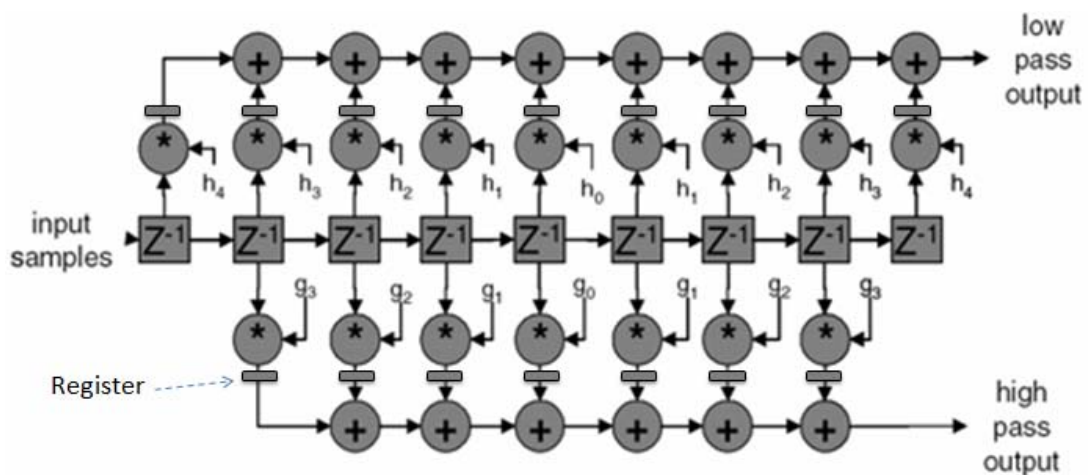


Fig. 2.1 Basic DWT architecture by 9/7-tap Daubechies FIR filter.

The basic 9/7-tap Daubechies architecture defined in Fig. 2.1 uses 14 adders, 16 multipliers and 9 registers. This architecture consumes high area cost for the parallel implementation. Thus, lifting algorithm was developed to reduce this area cost.

2.3. Lifting scheme for Discrete Wavelet Transform

The lifting-based wavelet transform presented by Sweldens [13] is a sort of architecture featured by a sequence of smaller filters which can be converted into a sequence of upper and lower triangular matrices. The sequence of smaller filters was achieved from breaking up the high-pass and low-pass wavelet filters [13]. There are some advantages of the lifting-based wavelet transform for DWT implementation when compares to the conventional convolution-based approach such as easily integer operations using less computation requirements and avoid the problems caused by the finite precision or rounding.

For lifting architecture, the traditional conditions (filtering) for reconstruction of signal [3] are given by

$$\begin{aligned} h(z)\tilde{h}(z^{-1}) + g(z)\tilde{g}(z^{-1}) &= 2, \\ h(z)\tilde{h}(-z^{-1}) + g(z)\tilde{g}(-z^{-1}) &= 0 \end{aligned} \quad (2.3)$$

where $h(z)$ is the Z-transform of the FIR filter h , for example, and *polyphase matrices* can be defined as

$$\tilde{P}(z) = \begin{bmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{g}_e(z) & \tilde{g}_o(z) \end{bmatrix}, \quad P(z) = \begin{bmatrix} h_e(z) & g_e(z) \\ h_o(z) & g_o(z) \end{bmatrix} \quad (2.4)$$

$$\begin{bmatrix} y_L(z) \\ y_H(z) \end{bmatrix} = \tilde{P}(z) \begin{bmatrix} x_e(z) \\ z^{-1}x_o(z) \end{bmatrix}, \quad \begin{bmatrix} x_e(z) \\ z^{-1}x_o(z) \end{bmatrix} = P(z) \begin{bmatrix} y_L(z) \\ y_H(z) \end{bmatrix} \quad (2.5)$$

where h_e and h_o contain the even and the odd coefficients of the FIR filter h respectively. Normally $P(z)$ is named as *dual* of $\tilde{P}(z)$. The expression (2.6) shown wavelet transform in terms of the polyphase matrix for the forward and inverse

DWT respectively, where $\tilde{s}_i(z)$ and $\tilde{t}_i(z)$ (for $1 \leq i \leq m$) are Laurent polynomials of lower orders, K is a constant and act as a scaling factor. Computation of the upper and low triangular matrix are known as *primal lifting* and *dual lifting*, respectively [3, 13]. Normally, these two basic lifting steps are called *update* and *predict*.

$$\tilde{P}(z) = \left\{ \prod_{i=1}^m \begin{bmatrix} 1 & \tilde{s}_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \tilde{t}_i(z) & 1 \end{bmatrix} \right\} \begin{bmatrix} K & 0 \\ 0 & \frac{1}{K} \end{bmatrix} \quad (2.6)$$

where K is a constant. The two types of lifting schemes are shown in Fig. 2.2.

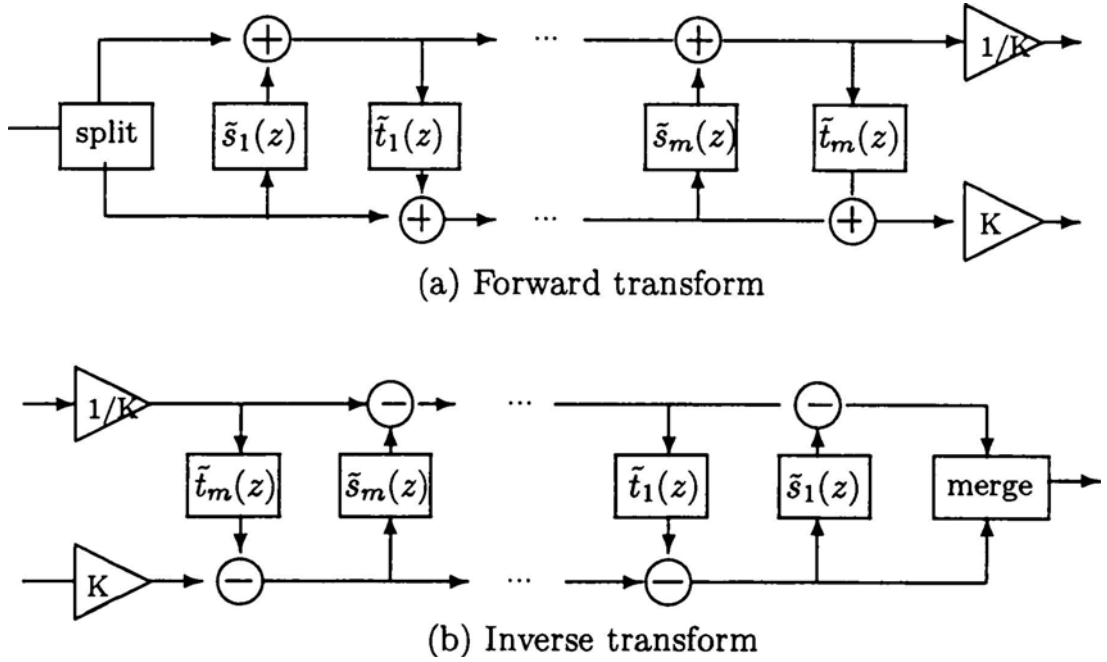


Fig. 2.2 Lifting based forward and inverse DWT.

As an example (5, 3) filter, with $\tilde{h} = (-\frac{1}{8}, \frac{1}{4}, \frac{3}{4}, \frac{1}{4}, -\frac{1}{8})$ and $\tilde{g} = (-\frac{1}{2}, 1, -\frac{1}{2})$. The polyphase matrix of this filter bank can be derived as

$$\tilde{P}(z) = \begin{bmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{g}_e(z) & \tilde{g}_o(z) \end{bmatrix} = \begin{bmatrix} -\frac{1}{8}z^{-1} + \frac{3}{4} - \frac{1}{8}z & \frac{1}{4} + \frac{1}{4}z \\ -\frac{1}{2}z^{-1} - \frac{1}{2} & 1 \end{bmatrix}$$

then, can be factorized as

$$\tilde{P}(z) = \begin{bmatrix} 1 & \frac{1}{4}(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\frac{1}{2}(1+z^{-1}) & 1 \end{bmatrix}$$

If the number of samples are starting from 0, we can consider the even and odd terms as the samples of lowpass and highpass subband respectively. Hence, we can interpret the above matrices in the time domain as $y_{2i+1} = b(x_{2i} + x_{2i+2}) + x_{2i+1}$ and $y_{2i} = a(y_{2i+1} + y_{2i+3}) + x_{2i}$, where $a = -\frac{1}{2}$ and $b = \frac{1}{4}$, $0 \leq i \leq \frac{N}{2}$.

The (9, 7) wavelet filter has been proposed in JPEG2000 standard, the polyphase can be factorized [3] as

$$\bar{P}(z) = \begin{bmatrix} 1 & a(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ b(1+z) & 1 \end{bmatrix} \begin{bmatrix} 1 & c(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ d(1+z) & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & \frac{1}{K} \end{bmatrix}$$

where $a = -1.586134342$, $b = -0.05298011854$, $c = 0.8829110762$, $d = -0.4435068522$, and $K = 1.149604398$.

Therefore, the forward transform for (5, 3) and (9, 7) filters can be represented as $Y_{(5,3)} = XM_1M_2$ and $Y_{(9,7)} = XM_1M_2M_3M_4$ respectively, where

$$M_1 = \begin{bmatrix} 1 & a & 0 & . & . & . & . & . & . \\ 0 & 1 & 0 & 0 & . & . & . & . & . \\ 0 & a & 1 & a & 0 & . & . & . & . \\ . & 0 & 0 & 1 & 0 & 0 & . & . & . \\ . & . & 0 & a & 1 & a & 0 & . & . \\ . & . & . & 0 & 0 & 1 & 0 & 0 & . \\ . & . & . & . & 0 & a & 1 & a & 0 \\ . & . & . & . & . & 0 & 0 & 1 & 0 \\ 0 & . & . & . & . & . & 0 & a & 1 \end{bmatrix}, \quad M_2 = \begin{bmatrix} 1 & 0 & 0 & . & . & . & . & . & . \\ 0 & 1 & b & 0 & . & . & . & . & . \\ 0 & 0 & 1 & 0 & 0 & . & . & . & . \\ . & 0 & b & 1 & b & 0 & . & . & . \\ . & . & 0 & 0 & 1 & 0 & 0 & . & . \\ . & . & . & 0 & b & 1 & b & 0 & . \\ . & . & . & . & 0 & 0 & 1 & 0 & 0 \\ . & . & . & . & . & 0 & b & 1 & 0 \\ 0 & . & . & . & . & . & 0 & 0 & 1 \end{bmatrix}$$

$$M_3 = \begin{bmatrix} 1 & c & 0 & . & . & . & . & . & . \\ 0 & 1 & 0 & 0 & . & . & . & . & . \\ 0 & c & 1 & c & 0 & . & . & . & . \\ . & 0 & 0 & 1 & 0 & 0 & . & . & . \\ . & . & 0 & c & 1 & c & 0 & . & . \\ . & . & . & 0 & 0 & 1 & 0 & 0 & . \\ . & . & . & . & 0 & c & 1 & c & 0 \\ . & . & . & . & . & 0 & 0 & 1 & 0 \\ 0 & . & . & . & . & . & 0 & c & 1 \end{bmatrix}, \quad M_4 = \begin{bmatrix} 1 & 0 & 0 & . & . & . & . & . & . \\ 0 & 1 & d & 0 & . & . & . & . & . \\ 0 & 0 & 1 & 0 & 0 & . & . & . & . \\ . & 0 & d & 1 & d & 0 & . & . & . \\ . & . & 0 & 0 & 1 & 0 & 0 & . & . \\ . & . & . & 0 & d & 1 & d & 0 & . \\ . & . & . & . & 0 & 0 & 1 & 0 & 0 \\ . & . & . & . & . & 0 & d & 1 & 0 \\ 0 & . & . & . & . & . & 0 & 0 & 1 \end{bmatrix}$$

Several VLSI architectures have been designed ([17] [18] [20] [21]) for implementation of the lifting steps. These architectures range from highly parallel architectures to programmable DSP-based architectures to folded architectures. We present systematic derivations of some of Daubechies architectures in the following sections.

2.4. Existing DWT Implementation

2.4.1 DWT VLSI Design by using Daubechies

Daubechies-based wavelet transform is the first generation of DWT VLSI design, implemented by using the Daubechies biorthogonal wavelet coefficients [1]. Some papers presented and improved this architecture.

[14] presented a simple concept of low-complexity, efficient 9/7 wavelet filters VLSI structure followed the JPEG2000 standard. The performance of a hardware implementation of the 9/7 filter bank depends on the accuracy of coefficients representation. This architecture shows substantial complexity reduction with as good performance as the 9/7-tap implementation, shown in Fig 2.3. Almost 60% of multipliers was used when compares with the basic-Daubechies architecture in Fig 2.1. Furthermore, these multipliers were replaced by shifts which is shown in Fig 2.4. However, it still be a direct FIR architecture.

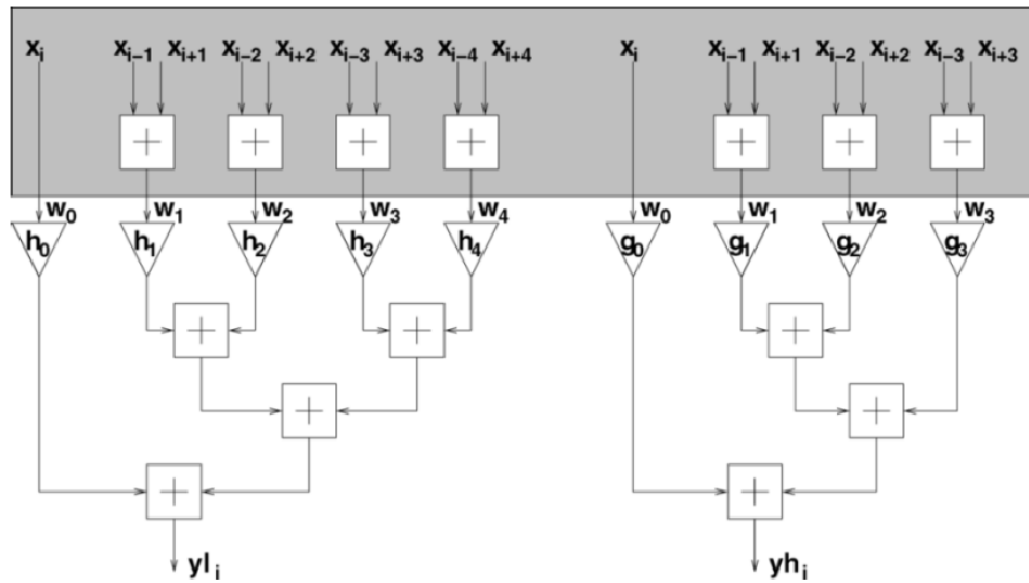


Fig. 2.3 Fast 9/7 direct implementation [14].

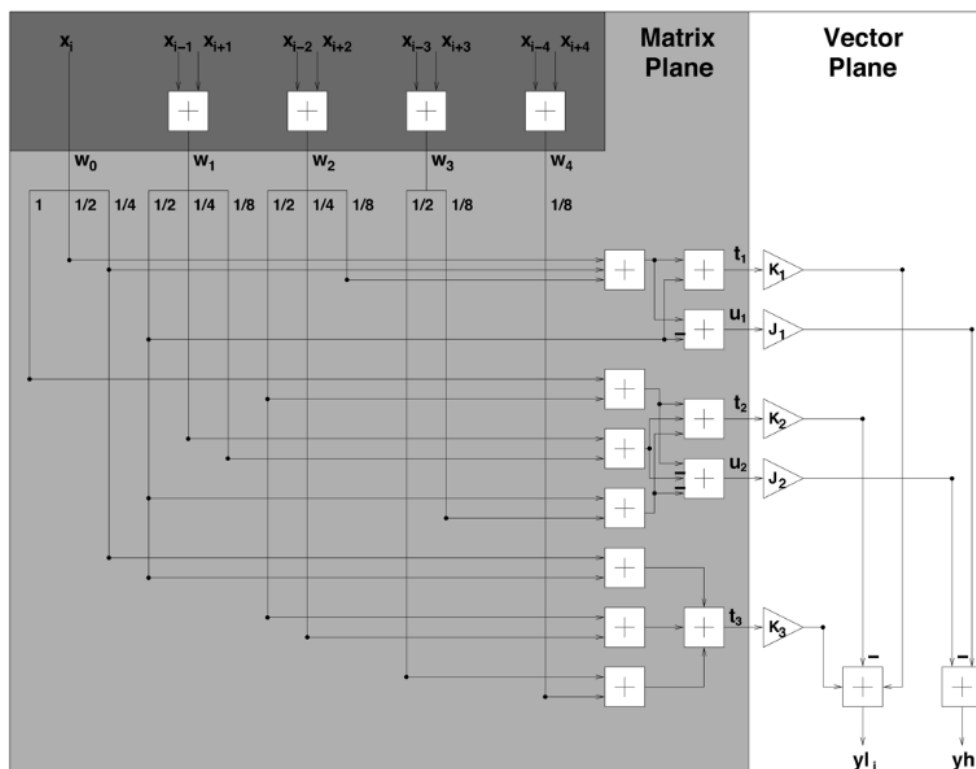


Fig. 2.4 Fast modified 9/7 direct implementation [14].

[15] presented a DWT filter bank which is constructed of simple architectures by using Daubechises coefficients. It is a transposed form FIR filter, as

shown in Fig 2.5. The input registers were reduced and removed complex control for users, but more registers were used between the multiplier block and the adder block.

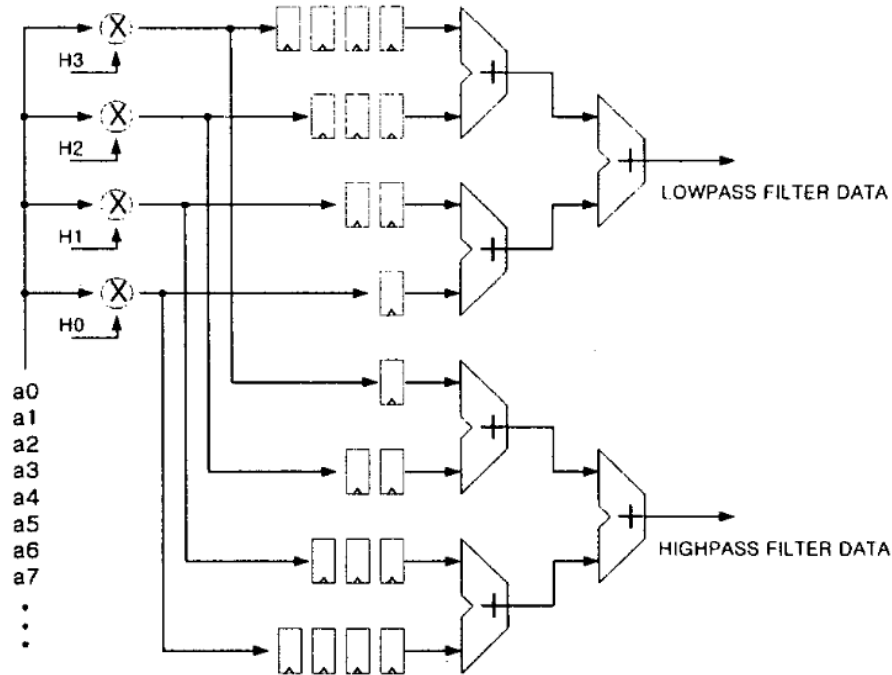


Fig. 2.5 DWT filter bank of Transposed form FIR [15].

The architecture features parallel and pipelined computation and high throughput shown in Fig 2.6 [16]. This architecture is similar to my design (Fig. 3.8), but only for using 4-tap Daubechies filters. However this high-efficiency architectures for the even and odd parts of 1-D convolution is 100% hardware-utilization, multiplierless, regular structure, simple control flow and high scalability. The multiplier can be replaced by a carry-save adder (CSA), and three hardware shifters in processing element (PE). The shift and adder circuit replacing the multiplier is shown in Fig 2.7. In this architectures, multiplications are obtained by shifts and additions after approximating the coefficients in both binary recoded format (BBRF). The replaced multiplier can reduce power dissipation by m in comparison to the conventional architectures in m -bit operands [16].

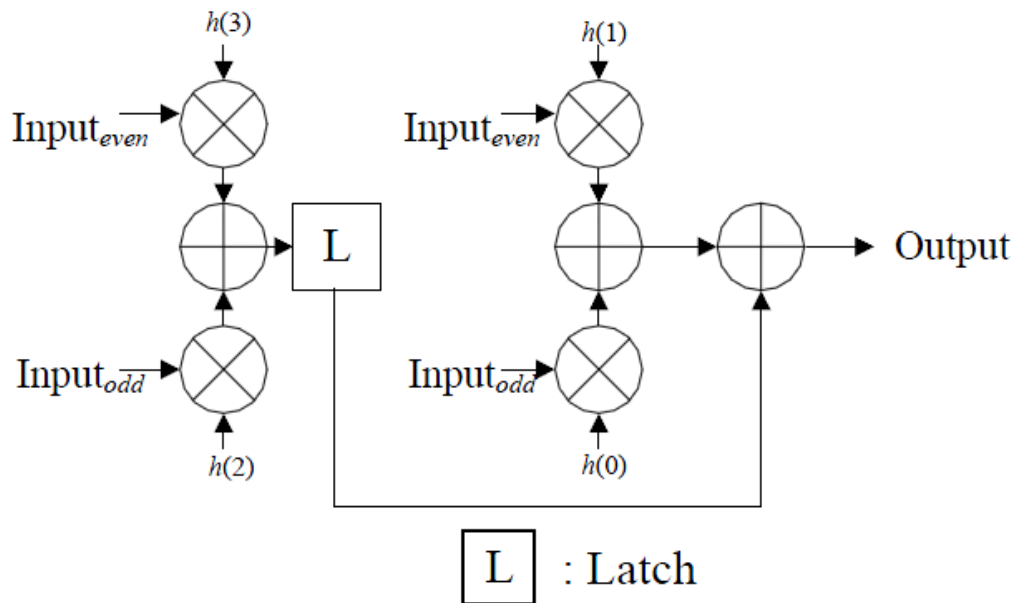


Fig. 2.6 The horizontal low-pass filter for 1-D DWT [16].

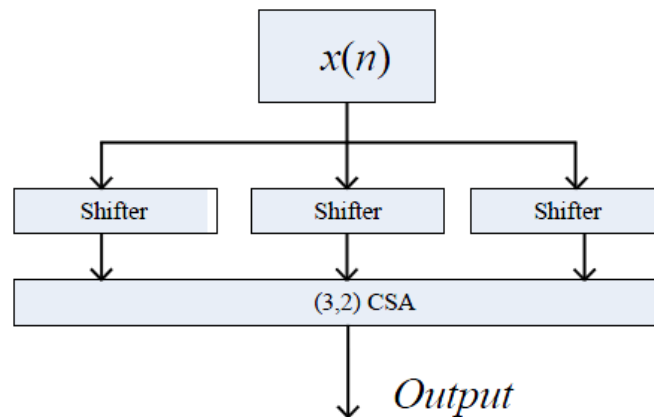


Fig. 2.7 The shift and adder circuits replacing the multiplier [16].

2.4.2 DWT VLSI Design by using Lifting Scheme

2.4.2.1 Lifting Architectures for 1-D DWT [17]

The data dependencies in the lifting scheme can be explained with the help of an example of DWT filtering with four factors (or four lifting steps). The four lifting steps correspond to four stages as shown in Fig. 1.4 (a). The intermediate

results generated in the first two stages for the first two lifting steps are subsequently processed to produce the high-pass (HP) outputs in the third stage, followed by the low-pass (LP) outputs in the fourth stage. (9,7) filter is an example of a filter that requires four lifting steps. For the DWT filters requiring only two factors, such as the (5,3) filter, the intermediate two stages can simply be bypassed.

2.4.2.2 Direct Mapped Architecture [17]

A direct mapping of the data dependency diagram into a pipelined architecture was proposed by Liu et al. in [17] as depicted in Fig. 2.8. The architecture is designed with 8 adders (A1-A8), 4 multipliers (M1-M4), 6 delay elements (D) and 8 pipeline registers (R). There are two input lines to the architecture: one that inputs even samples $\{X_{2i}\}$, and the other one that inputs odd samples $\{X_{2i+1}\}$. There are four pipeline stages in the architecture. In the first pipeline stage, adder A1 computes $X_{2i} + X_{2i-2}$ and adder A2 computes $a(X_{2i} + X_{2i-2}) + X_{2i-1}$. The output of A2 corresponds to the intermediate results generated in the first stage of Fig. 2.2(a). The output of adder A4 in the second pipeline stage corresponds to the intermediate results generated in the second stage of Fig. 2.2(a). Continuing in this fashion, adder A6 in the third pipeline stage produces the high-pass output samples, and adder A8 in the fourth pipeline stage produces the low-pass output samples. For Lifting schemes that require only 2 lifting steps, such as the (5,3) filter, the last two pipeline stages need to be bypassed causing the hardware utilization to be only 50% or less. Also, for a single read port memory, the odd and even samples are read serially in alternate clock cycles and buffered. This slows down the overall pipelined architecture by 50% as well. A similar pipelined architecture for the (9,7) wavelet has been proposed by Jou et al. in [19].

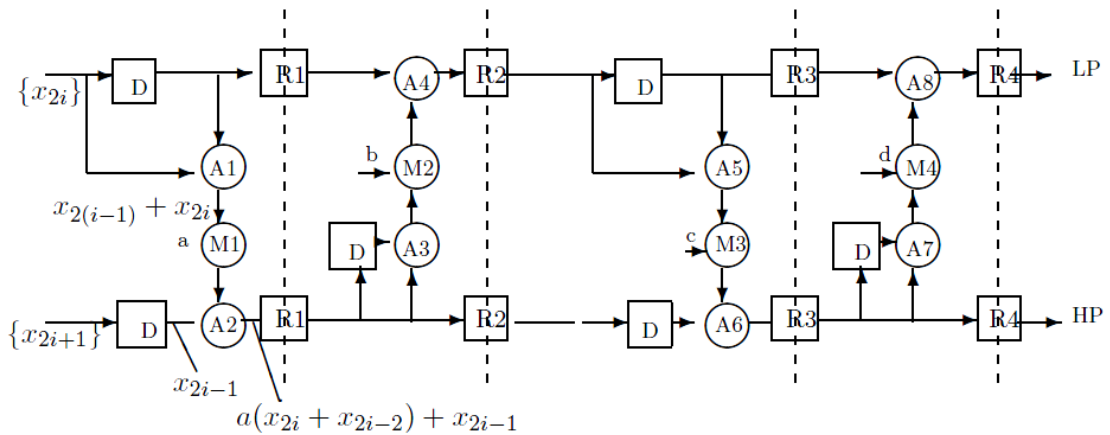


Fig. 2.8 The direct mapped architecture [17].

2.4.2.3 Folded Architecture [18]

The pipelined architecture in Fig. 2.8 can be further improved by carefully folding the last two pipeline stages into the first two stages as shown in Fig. 2.9. The architecture proposed by Lian, et al. in [18] consists of two pipeline stages, with three pipeline registers, R1, R2 and R3. In the (9,7) type filtering operation, intermediate data (R3) generated after the first two lifting steps (phase1) are folded back to R1 (as shown in Fig. 2.9) for computation of the last two lifting steps (phase2). The architecture can be reconfigured so that computation of the two phases can be interleaved by selection of appropriate data by the multiplexors. As a result, two delay registers (D) are needed in each lifting step in order to properly schedule the data in each phase. Based on the phase of interleaved computation, the coefficient for multiplier M1 is either a or c , and similarly the coefficient for multiplier M2 is b or d . The hardware utilization of this architecture is always 100%. Note that for the (5,3) type filter operation, folding is not required.

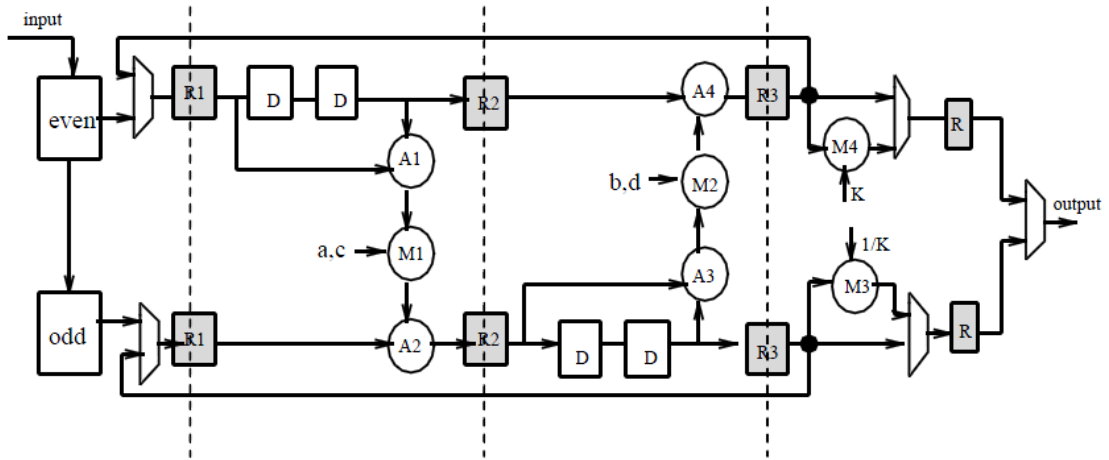


Fig. 2.9 The folded architecture in [18].

2.4.2.4 MAC Based Programmable Architecture [20]

A programmable architecture that implements the data dependencies represented in Fig. 2.2(a) using four MACs (Multiply and Accumulate) and nine registers has been proposed by Chang et al. in [20]. The algorithm is executed in two phases as shown in Fig. 2.10. The data-flow of the proposed architecture can be explained in terms of the register allocation of the nodes. The computation and allocation of the registers in phase 1 are done in the following order

$$R0 \leftarrow X_{2i-1}; R2 \leftarrow X_{2i};$$

$$R3 \leftarrow R0 + a(R1 + R2);$$

$$R4 \leftarrow R1 + b(R5 + R3);$$

$$R8 \leftarrow R5 + c(R6 + R4);$$

Similarly, the computation and register allocation in phase 2 are done in the following order

$$R0 \leftarrow X_{2i-1}; R2 \leftarrow X_{2i};$$

$$R5 \leftarrow R0 + a(R2 + R1);$$

$$R6 \leftarrow R2 + b(R3 + R5);$$

$$R7 \leftarrow R3 + c(R4 + R6);$$

$$\text{Output(LP)} \leftarrow R4 + d(R8 + R7); \quad \text{Output(HP)} \leftarrow R7$$

As a result, two samples are input per phase and two samples (LP and HP) are output at the end of every phase. For 2D DWT implementation, the output samples are also stored into a temporary buffer for filtering in the vertical dimension.

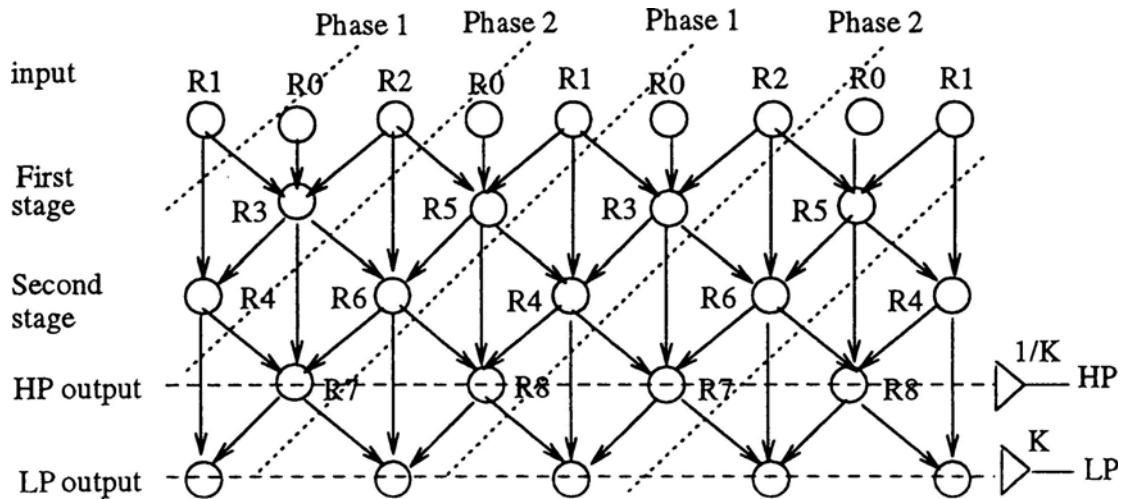


Fig. 2.10 Data-flow and register allocation of the MAC based architecture in [20].

2.4.2.5 Flipping Architecture [21]

While conventional lifting-based architectures require fewer arithmetic operations, they sometimes have long critical paths. For instance, the critical path of the lifting-based architecture for the (9,7) filter is $4T_m + 8T_a$ while that of the convolution implementation is $T_m + 4T_a$. One way to improve this is by pipelining which results in a significant increase in the number of registers. For instance, to pipeline the lifting-based (9,7) filter such that the critical path is $T_m + 2T_a$, 6 additional registers are required.

Recently, Huang et al. [21] proposed a very efficient way of solving the timing accumulation problem. The basic idea is to remove the multiplications along the critical path by scaling the remaining paths by the inverse of the multiplier coefficients. Fig. 2.11(a)-(c) describes how scaling at each level can reduce the multiplications in the critical path. Fig. 2.11(d) further splits the three input addition

nodes into two 2-input adders. The critical path is now $T_m + 5T_a$. The minimum critical path of T_m can be achieved by 5 pipelining stages using 11 pipelining registers (not shown in the figure). Detailed hardware analysis of lossy (9,7), integer (9,7) and (6,10) filters have been included in [21]. Further more, since the flipping transformation changes the round-off noise considerably, techniques to address precision and noise problems have also been addressed in [21].

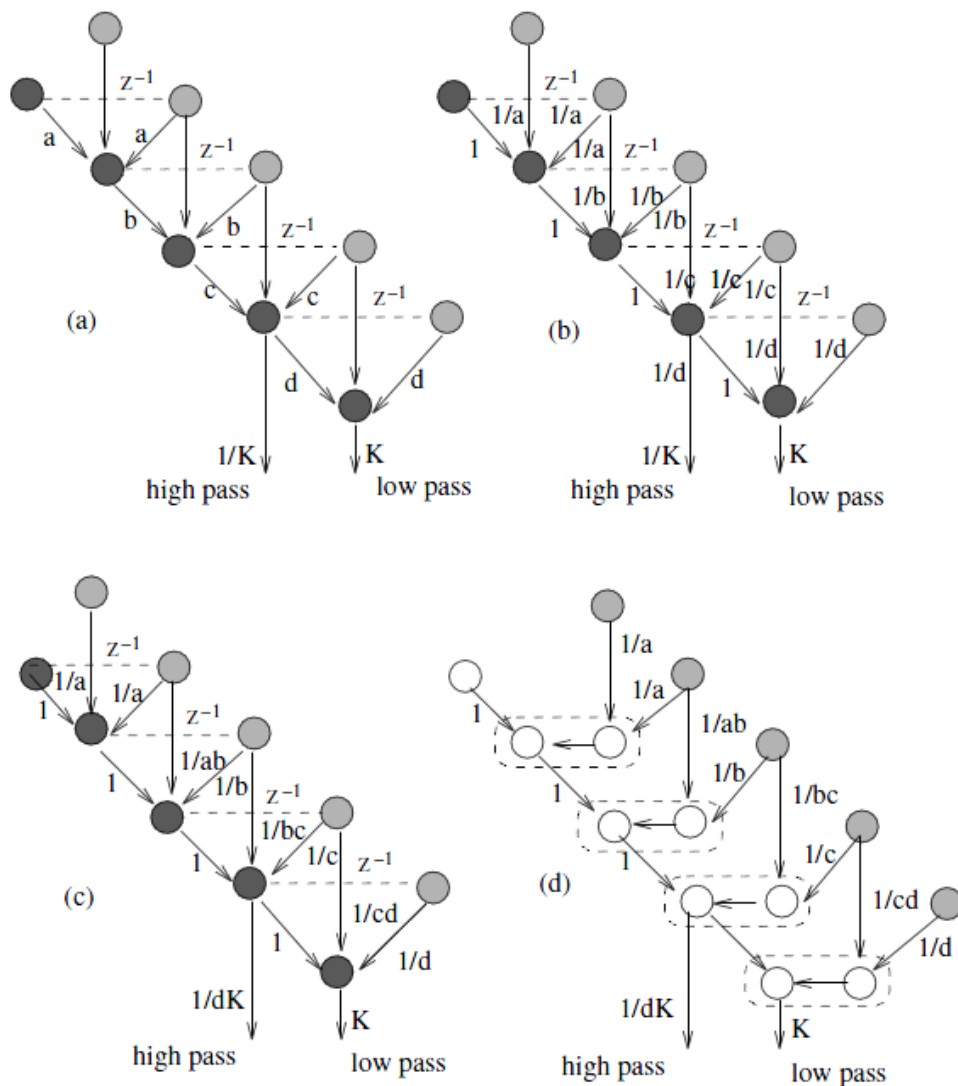
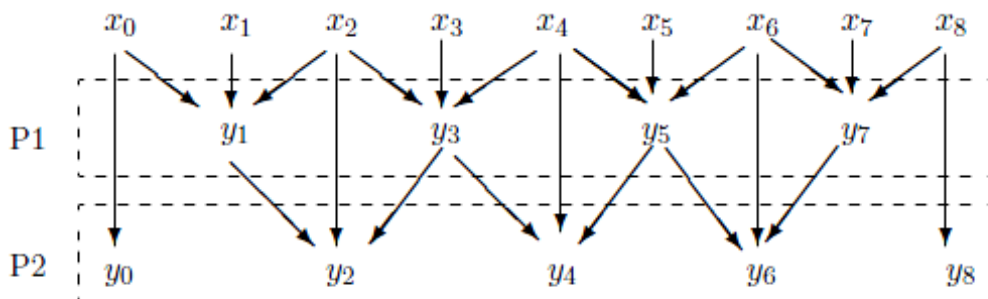


Fig. 2.11 A flipping architecture [21]. (a) Original architecture, (b)-(c) Scaling the coefficients to reduce the number of multiplications, (d) Splitting the three-input addition nodes to two-input nodes.

2.4.2.6 Generalized Architecture [22]

The architecture proposed by Andra et al. [22] is an example of a highly programmable architecture that can support a large set of filters. These include filters (5,3), (9,7), C(13,7), S(13,7), (2,6), (2,10), and (6,10). Since the data dependencies in the filter computations can be represented by at most four stages, the architecture consists of four processors, where each processor is assigned computations of one stage. Fig. 2.12(a) describes the assignment of computation to two processors, P1 and P2, for the (5,3) filter which can be represented by two stages.



(a) Processor assignment for the (5,3) filter

Cycle	Processor 1 (P1)			Processor 2 (P2)		
	Adder1	Shifter	Adder2	Adder1	Shifter	Adder2
1	–	–	–	–	–	–
2	$x_0 + x_2$	–	–	–	–	–
3	$x_2 + x_4$	R1	–	–	–	–
4	$x_4 + x_6$	R1	$Rs - x_1 = y_1$	–	–	–
5	$x_6 + x_8$	R1	$Rs - x_3 = y_3$	–	–	–
6	–	R1	$Rs - x_5 = y_5$	y_1, y_3	–	–
7	–	–	$Rs - x_7 = y_7$	y_3, y_5	R1	y_0
8	–	–	–	y_5, y_7	R1	$Rs + x_2$
9	–	–	–	–	R1	$Rs + x_4$
10	–	–	–	–	–	$Rs + x_6$

(b) Partial schedule for the (5,3) filter implementation

Fig. 2.12 Processor assignment and partial schedule for the (5,3) filter implementation in the Generalized architecture in [22].

The processor architecture consists of adders, multipliers and shifters that are interconnected in a manner that would support the computational structure of the specific filter. Fig. 2.13 describes the processor architectures for the (5,3) filter and the (9,7) filter. While the (5,3) filter architecture consists of two adders, and a shifter, the (9,7) filter architecture consists of two adders and a multiplier. Fig. 2.12(b) describes part of the schedule for the (5,3) filter. The schedules are generated by mapping the data dependency graph onto the resource-constrained architecture. It is assumed that the delays of each adder, shifter and the multiplier are 1, 1 and 4 time units respectively. For example (Fig. 2.11(b)), Adder 1 of P1 adds the elements (x_0, x_2) in the 2nd cycle and stores the sum in register R1. The shifter reads this sum in the next cycle (3rd cycle), carries out the required number of shifts (one right shift as $a = -0.5$) and stores the data in register Rs. The second adder (Adder2) reads the value in Rs and subtracts the element x_1 to generate y_1 in the next cycle. To process $N=9$ data, the P1 processor takes four cycles. Adder 1 in P2 processor starts computation in the sixth cycle. The gaps in the schedules for P1 and P2 are required to store the zeroth element of each row.

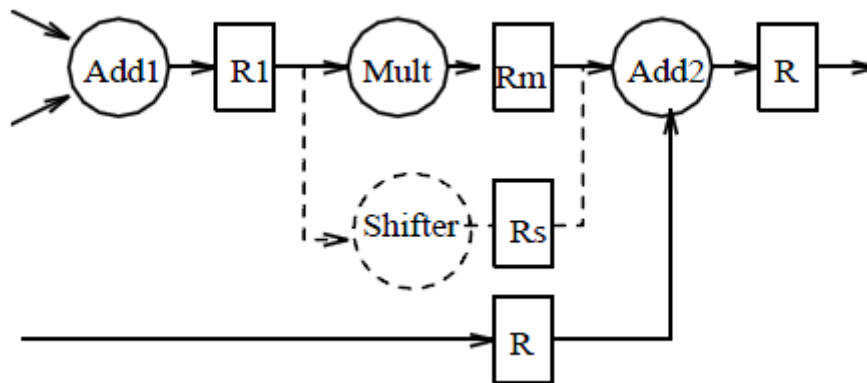


Fig. 2.13 Processor architecture for the (5,3) and (9,7) filters in [22].

2.4.2.7 Recursive Architecture [24]

Most of the traditional DWT architectures compute the i th level of decomposition upon completion of $(i-1)$ th level of decomposition. However in multiresolution DWT, the number of samples to be processed in each level is always

half of the size in the previous level. Thus it is possible to process multiple levels of decomposition simultaneously. This is the basic principle of a *recursive architecture* that was first proposed for a convolution based DWT in [23] and applied for lifting based DWT in [24] [25]. Here computations in higher levels of decomposition is initiated as soon as enough intermediate data in low-frequency subband is available for computation. The proposed architecture for a 3-level decomposition of an input signal using Daubechies-4 DWT is shown in Fig. 2.14.

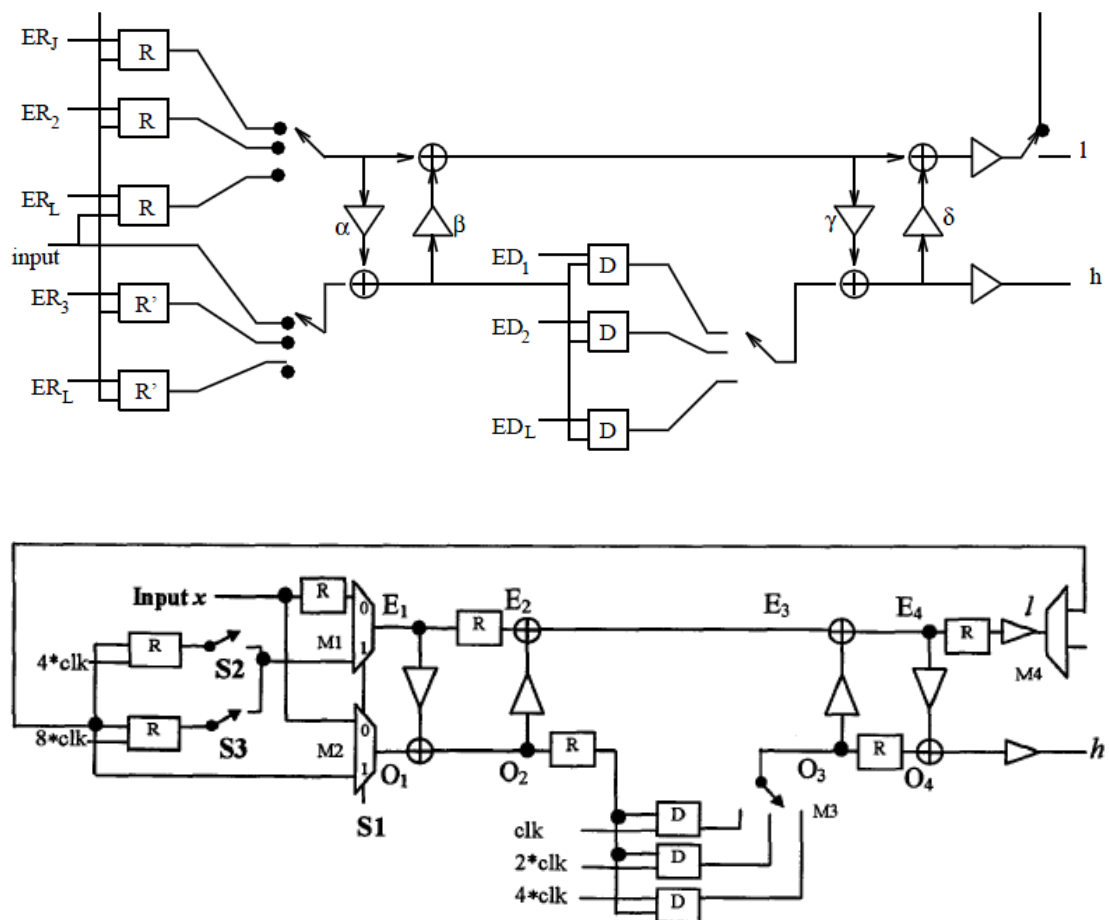


Fig. 2.14 The recursive architecture in [24].

The basic circuit elements used in this architecture are delay elements, multipliers and MAC units which are in turn designed using a multiplier, an adder and two shifters. The multiplexors M1 and M2 select the even and odd samples of the input data as needed by the lifting scheme. S1, S2 and S3 are the control signals for data flow of the architecture. The select signal (S1) of the multiplexors is set to 0 for

the first level of computation and is set to 1 during the second or third level computation. The switches S2 and S2 select the input data for the second and third level of computation. The multiplexor M3 selects the delayed samples for each level of decomposition based on the clocked signals show in Fig. 2.14.

A recursive architecture for 1D implementation of the (5,3) filter has been proposed in [26]. The architecture has hardware complexity identical to [25] but is claimed to be more regular. The topology is similar to a scan chain, and thus can be easily modified to support testable scan-based designs.

2.4.2.8 Summary of Lifting Architecture

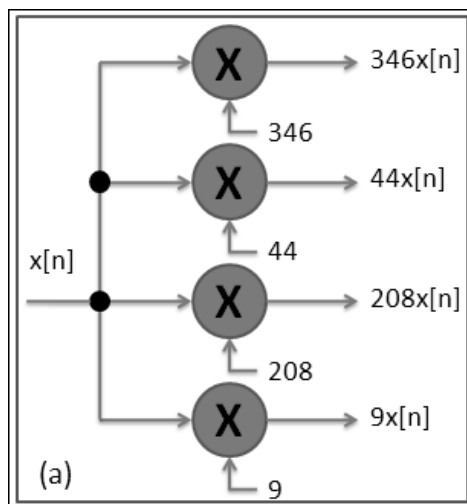
In fact, all of existing lifting architecture didn't improve the method of computation, the main idea of these lifting architectures essentially just used basic-lifting architecture with FIFO method or buffer memory control to improve. Hence, any one of the existing lifting architecture has to use lower speed for reduce resource and vice versa, in another word, they just via modified lifting architecture to achieve different choice for making a balance between speed and resource for different application requirement.

2.5. RAG for High-Performance

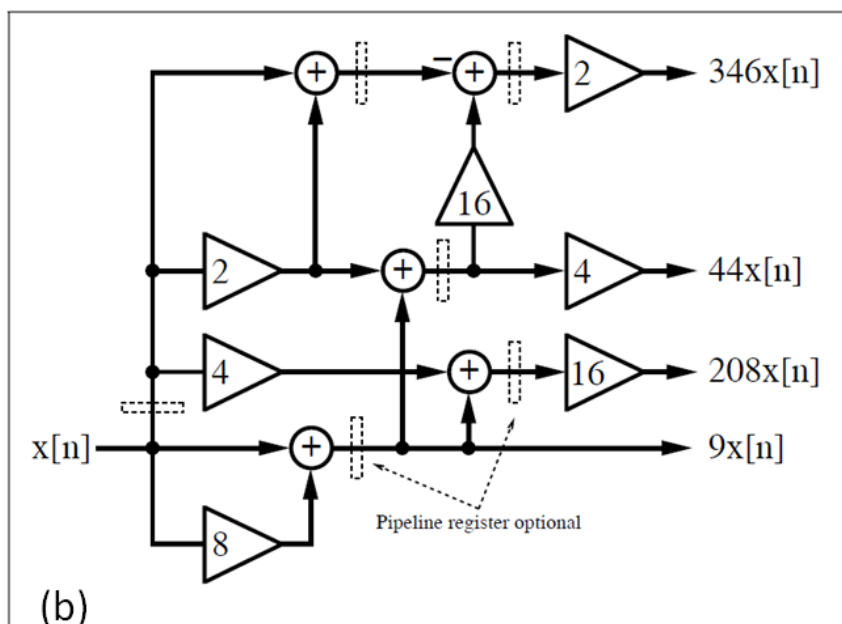
The transposed FIR enjoys, in the case of a constant coefficient filter, two additional improvements: multiple uses of the repeated coefficients using the reduced adder graph (RAG) algorithm [4], and pipeline adders using a carry-save adder.

Fig. 2.15 illustrates the RAG algorithm. Fig. 2.15 (a) is the multiplier block of transposed filter which has four nonzero coefficients, namely $f[0]$, $f[1]$, $f[3]$ and $f[5]$, which are 346, 208, -44, and 9. For cost estimation we convert the decimal values (index 10) into binary representations (index 2) for coefficients. Fig. 2.15 (b)

shows the resulting reduced adder graph. It used 7 shifts and 5 adders instead of 4 multipliers.



(a)



(b)

$f[k]$	$f[k]$	Cost
$f[0] = 346_{10} = 2 \times 173 = 101011010_2$		4
$f[1] = 208_{10} = 2^4 \times 13 = 11010000_2$		2
$f[3] = -44_{10} = -2^2 \times 11 = -101100_2$		2
$f[5] = 9_{10} = 3^2 = 1001_2$		1
Total		9

Fig. 2.15 Realization of F6 using RAG algorithm [27].

2.6. Lifting 2-D DWT recursive Architecture [28]

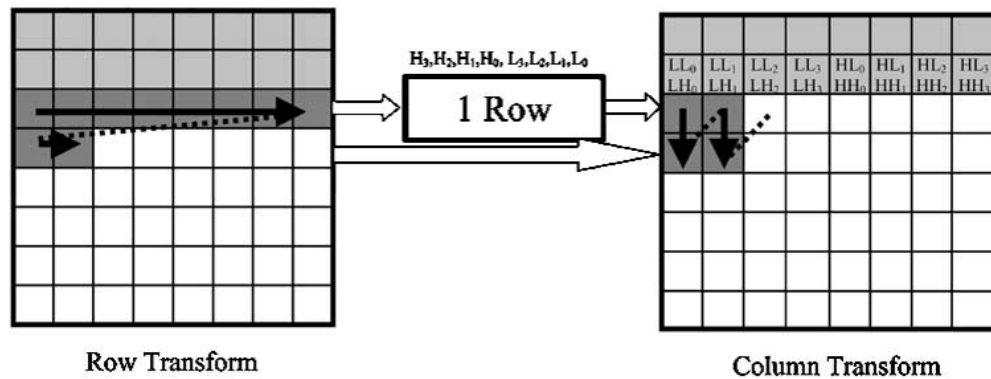


Fig. 2.16 Calculation sequence of the 2-D RA.

The basic strategy of the 2-D recursive architecture is the same as that of its 1-D counterpart. The calculations of all DWT stages are interleaved to increase the hardware utilization. Within each DWT stage, processing sequences of each DWT stage is shown in Fig. 2.16. The image is scanned into the row processor in a raster format, and the first horizontal DWT is immediately started. The resulting high- and low-frequency DWT coefficients of the odd lines are collected and pushed into two first-in first-out (FIFO) registers or two memory banks. The separate storages of the high- and low-frequency components produce a more regular data flow and reduce the required output switch operations, which in turn consume less power. The DWT coefficients of the even lines are also rearranged into the same sequence and are directly sent to the column processor, together with the output of the FIFO. The column processor starts calculating the vertical DWT in a zigzag format after one row's delay.

The block diagram of the lifting 2-D DWT architecture is shown in Fig. 2.17. The FPGA implements a row processor (RP), a column processor (CP), and a local memory module (MEM) used to buffer results between the RP and CP. The 2-D DWT is computed in row-column fashion (i.e., the DWT is carried out on the rows first and then on the columns). The image, which is stored in external memory, is read to the FPGA in row-by-row order. The row processor performs horizontal filtering to the rows and writes the approximation, a , and detail, d , coefficients to the local

memory. Once a sufficient number of rows have been processed, the column processor starts vertical filtering; it fetches coefficients from the local memory and generates four subbands, aa , da , dd , and ad , as shown in Fig. 2.18. The four subbands are written back to the external memory, again in row-wise order.

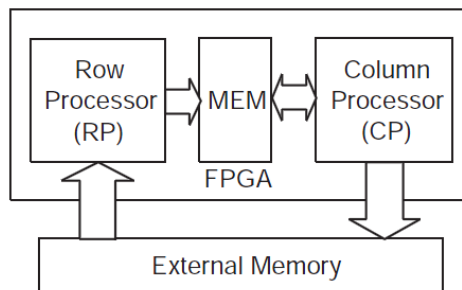


Fig. 2.17 System architecture [28].

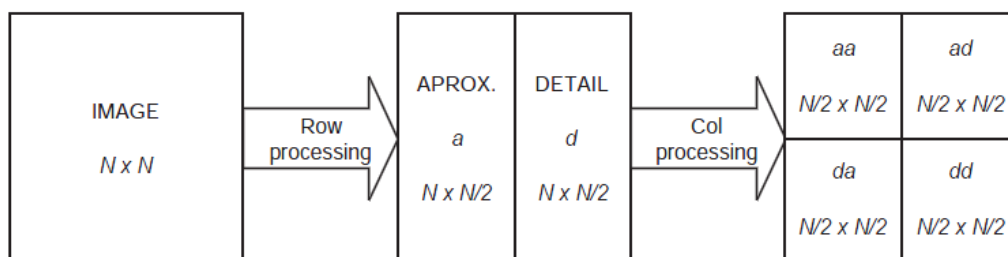


Fig. 2.18 A one-level 2-D DWT [28].

Multiple levels are performed on this architecture in a non-interleaved fashion, with results between levels stored in the external memory. For each of the higher levels, an approximation subband is read from external memory and four higher level subbands are generated using the same computing modules. The operation continues until the desired level is finished.

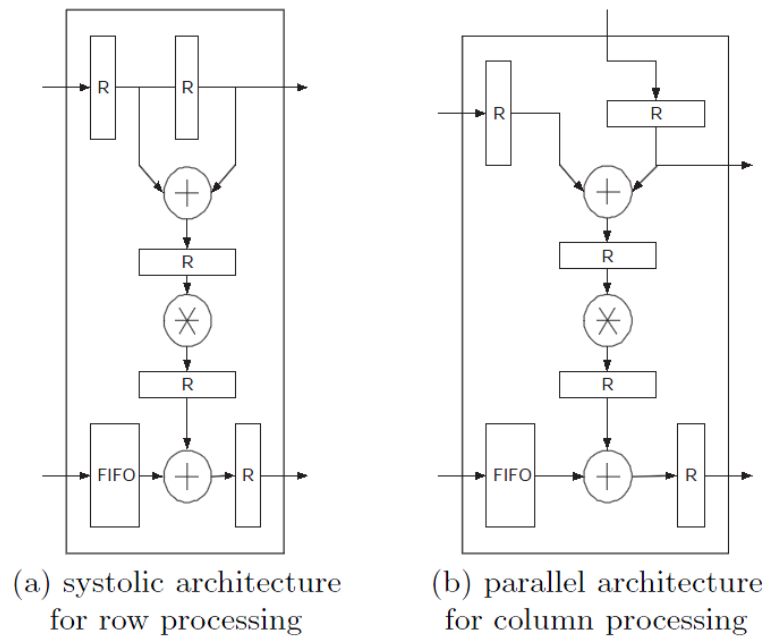


Fig. 2.19 Implementation of a Lifting step [28].

Row and column processing require two slightly different implementations of the lifting step. During row processing, the pixels of an image are read from external memory in a row-wise fashion, and needed in a row-wise order, making a systolic architecture of Fig. 2.19(a), which takes its inputs sequentially, ideally suitable. On the other hand, after row processing the pixels are stored in an embedded memory until enough column data is available to proceed with column processing. The inputs are best fed into the column processor in parallel, leading to the parallel architecture of Fig. 2.19(b). The first-in-first-out (FIFO) buffers shown in Fig. 2.19 are used to compensate for the latency of the pipelined multipliers within the lifting steps.

Lifting Row Processor (RP): Implementation of the row processor (RP) is straightforward; a functional block diagram is shown in Fig. 2.19. It consists of six computing modules: P1, U1, P2, U2, G1 and G2. Each lifting step (P1, U1, P2 and U2) is implemented using the systolic architecture of Fig. 2.20, and the steps are cascaded to build the whole lifting structure. At each clock, P1 takes a pair of coefficients, even and odd, and produces a pair of outputs that is then fed to the next module U1. Similarly U1 feeds P2, which feeds U2. Finally, the outputs of U2 are scaled through

G1 and G2, which are implemented as one-tap filters. The result is the one-dimensional DWT of the rows.

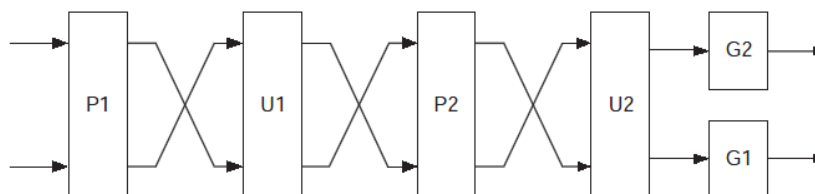


Fig. 2.20 Architecture of the row processor [28].

Lifting Column Process (CP): Coefficient exit the row processor in row-wise order, and are stored in an embedded memory until enough rows are buffered so that column processing can begin. When using convolution 9/7 wavelet filters, nine rows must be buffered. The column processing is then done in row-wise manner; the column filter works on nine pixels of data from a single column at a time. At each clock, the filter moves one column to the right. This architecture reduces the amount of buffer space required by exploiting a lifting structure. The lifting structure staggers when the column data is needed; although we still need nine pixels of data from a column to complete one wavelet coefficient, we need only three pixels to start the first lifting step. The remaining pixels are used in later clocks by later lifting steps. We shall see that in all, buffer space for only seven rows is required.

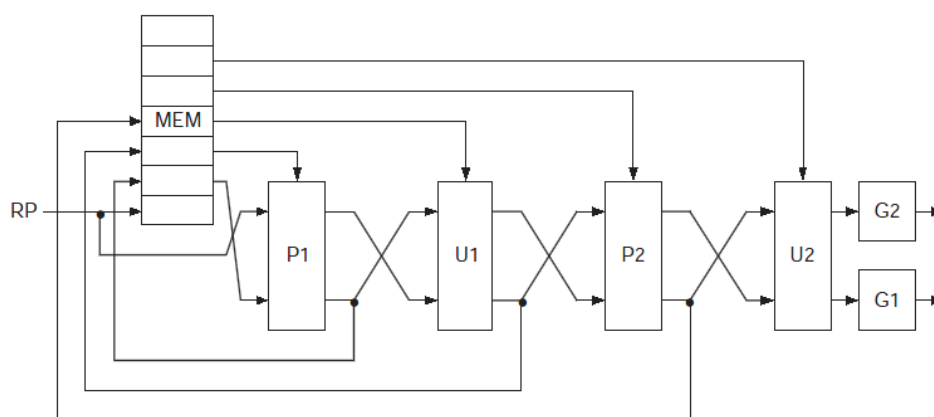


Fig. 2.21 Architecture of the column processor [28].

A block diagram of the column processor is shown in Fig. 2.21. The basic building blocks and their functions are the same as for the row processor except

that the parallel lifting step architecture is used. The lifting step computations are scheduled on the modules of the column processor so as to minimize the amount of embedded memory needed to buffer data between the row and column processing while keeping both the row and column processor continually busy. We can use ASAP scheduling: *each lifting step starts operation as soon as its inputs are available*. For the sake of simplicity of the figures, we assume that the latency of each computing module is one time unit. The schedule can be easily modified to account for additional latency.

The embedded memory is organized into seven lines of N words each. Each line has two parts: the first $N/2$ words are used to buffer approximation coefficients coming from the row processor, and the second $N/2$ words are used to buffer detail coefficients. At each clock, the row processor writes two new coefficients (one approximation and one detail). It moves from left to right along each line of the memory, and starts over at the beginning of the first line after it completes the seventh line.

As soon as the row processor (RP) completes the third row of coefficients, the first two lifting steps P1 and U1 start column processing. They travel left to right along the lines of memory, just behind the row processor. At each clock, P1 needs three inputs coming from a single column of the memory. One of the inputs is passed directly to it from the row processor. The other two are read from the embedded memory. P1 writes its result in the embedded memory, over the middle input (which will never be needed again). This result will be read by P2 at a later time. P1 also passed both the top input that it read and the result that it produced to U1. The other lifting steps follow along behind P1 in a similar way.

Chapter 3. The Proposed 1-D DWT Architecture

3.1. Expressions Formalization

The 1-D Daubechies architecture sequence is separately low-pass and high-pass filtered consisting of FIR filter coefficients. However, most common FIR filters are the linear time-invariant (LTI) filters. An FIR with constant coefficients is an LTI digital filter. The output of an FIR of order or length L , to an input time-series $x[n]$, and x is input signal, y is the convolved output, is given by a finite version of the convolution sum given in (3.1), namely:

$$y[n] = x[n] * f[n] = \sum_k x[k] f[n-k] \quad (3.1)$$

In contrast, the DWT expressions have two inputs, consist of even number $x[2n]$ and odd number $x[2n-1]$ of the input signal $x[n]$. Let a be the number of taps. The even-tap and odd-tap filters have different expressions as follows.

When the a of taps is odd number, the expression is as (3.2), set $a = 2b-1$, the n of $f[n]$, $n [b, -b]$, if $n [b, -b]$, $f[n]=0$.

$$y[n] = x[n] * f[n] = \sum_{k=0}^b (x[2k] f[b-2k] + x[2k+1] f[b-2k-1]) \quad (3.2)$$

When the a of taps is even number, the expression is as (3.3), set $a = 2b$, the n of $f[n]$, $n [b-1, -b-1]$, if $n [b-1, -b-1]$, $f[n]=0$.

$$y[n] = \sum_{k=0}^{b-1} (x[2k] f[b-2k-1] + x[2k+1] f[b-2k-2]) \quad (3.3)$$

For example number of a is 7, then b is 3, as Fig. 3.1, $f[n] = [g_3, g_2, g_1, g_0, g_{-1}, g_{-2}, g_{-3}]$, $x[2k]$ is even number of input signal, and $x[2k+1]$ is odd number of input signals. The implement function for plus sign of this expression is as the first row adders in Fig. 3.1. The implement function for the other additions is as the second row adders in Fig. 3.1.

3.2. 1-D Architecture for High/Low pass Filter

In this thesis, we propose an efficient pipeline architecture based on the Daubechies architecture. The proposed DWT is composed of two independent FIR filters: a high-pass FIR filter and a low-pass FIR filter. Each FIR filter is a two-stage pipeline with the fastest clock cycle of only either two adders delay or one multiplier delay. The propose DWT does not use the extra shift registers for storing the previous sampled data. It uses only two registers for the even input and the odd input. In additions, it also does not need extra shift registers for pipeline stages. Therefore, the proposed architecture is as simple as the Daubechies architecture, but uses less hardware resources. The architecture can achieve the higher speed than the lifting architecture because the clock cycle is one multiplier delay or two adders delay, not one multiplier delay plus two adders delay in the lifting architecture. The proposed architecture still has larger area for the implementation using fixed-point adders and multipliers, but occupies less area than Lifting architecture for the implementation by using the reduce adder graph (RAG) algorithm [4]. The proposed architecture can be further classified according to two structure types: even-tap filters and odd-tap filters. The multipliers occupation of even-tap filter is around 40% more than odd-tap filter.

The proposed design, named ZXY, has two steps as simple as Daubechies architecture as shown in Fig. 3.1. The operations are scheduled and assigned into two pipeline stages: a multiplier block and an adder block. In the first stage, the two input data are simultaneously multiplied by all coefficients. The second stage is the adder block which executes in order of formula on the right clock. Here, the multipliers and the adders, used in the high-pass filter and the low-pass filter, are the fixed-point format.

As an example, the 7-tap high-pass filter of the proposed 1-D 9/7-tap DWT is shown in Fig. 3.1. The 9-tap low-pass filter (not shown) has the similar structure. In the first step, the multipliers use the coefficients from 9/7 Daubechies filter. Because of the repeated coefficients (g_1 , g_2 , and g_3), the proposed 7-tap high-pass filter uses less three multipliers than the original Daubechies architecture of Fig. 2.1. Similarly, the 9-tap low-pass filter uses less four multipliers than the original

Daubechies architecture. The second step uses 6 adders which parallelly execute the data from the first- and second- pipeline stages. The latency of the first stage is one multiplier delay, and the latency of the second stage is two adder delays. Therefore, the pipeline clock is determined from the maximum delay of either one multiplier or two adders.

The proposed architecture has the similar structure as the general transposed FIR filter. However, the clock cycle of the pipeline version of the transposed FIR filter is constrained by one multiplier delay plus one adder delay as show in Fig. 2.1.

In Fig. 3.1, the defined registers R0-R7 have all of the results after every clock. The dataflow for 1-D DWT 7-tap high-pass filter is described in Table 3.1. R7 is output register for this high-pass filter, and has the same equation as (3.2). The circuit operates in pipeline manner with the latency of four clock cycles. The first sub-band output is produced at the forth clock cycle as shown in the column of R7. After the forth clock cycle, the pipeline produces the high-pass output at every clock. Here, the clock cycle is determined from the maximum of either 2 adder delays or 1 multiplier delay. Usually, the delay of a fixed-point multiplier is about 2 fixed-point adder delays. That means our pipeline is balanced, and the idle-time during pipeline stage is very small. Therefore, the high throughput performance can be achieved by the proposed architecture.

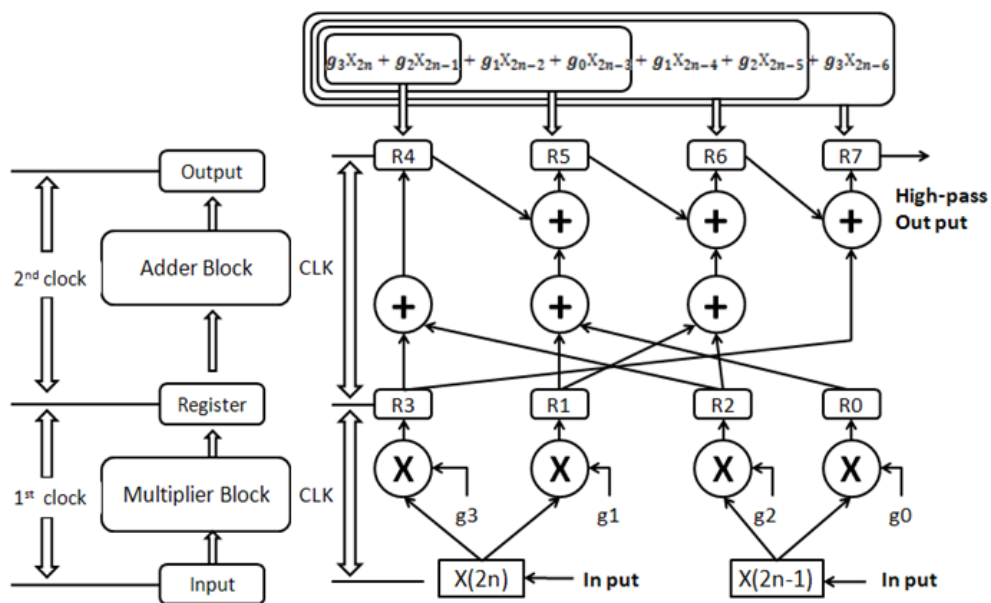


Fig. 3.1 The proposed 1-D DWT architecture for 7-tap high-pass filter.

The proposed architecture provides two options to tradeoff between area and speed. The structure shown in Fig. 3.1 is good for horizontal filters for 2-D DWT that needs high speed performance. There still have another option for vertical filters, for example 7-tap filter, just delete registers R0-R3 for area reduction, but the clock cycle will be extended, the detail shown in Table 3.1.

The control circuit for the proposed architecture is very simple. Observe that no control signal is needed in the two-stage pipelined FIR in Fig. 3.1. Only a global clock signal is used for all registers.

Table 3.1 The dataflow for the ZXY DWT 7-taps high-pass filter, about Fig. 3.1.

<i>clk</i>	<i>R3</i>	<i>R2</i>	<i>R1</i>	<i>R0</i>	<i>(next clk)R4</i> <i>=R3+R2</i>	<i>(next clk) R5</i> <i>=R4+R1+R0</i>
1	g_3x_1	g_2x_2	g_1x_1	g_0x_2	0	0
2	g_3x_3	g_2x_4	g_1x_3	g_0x_4	$g_3x_1+g_2x_2$	$g_1x_1+g_0x_2$
3	g_3x_5	g_2x_6	g_1x_5	g_0x_6	$g_3x_3+g_2x_4$	$g_3x_1+g_2x_2+g_1x_3+g_0x_4$
4	g_3x_7	g_2x_8	g_1x_7	g_0x_8	$g_3x_5+g_2x_6$	$g_3x_3+g_2x_4+g_1x_5+g_0x_6$
5	g_3x_9	g_2x_{10}	g_1x_9	g_0x_{10}	$g_3x_7+g_2x_8$	$g_3x_5+g_2x_6+g_1x_7+g_0x_8$
6	g_3x_{11}	g_2x_{12}	g_1x_{11}	g_0x_{12}	$g_3x_9+g_2x_{10}$	$g_3x_7+g_2x_8+g_1x_9+g_0x_{10}$
7	g_3x_{13}	g_2x_{14}	g_1x_{13}	g_0x_{14}	$g_3x_{11}+g_2x_{12}$	$g_3x_9+g_2x_{10}+g_1x_{11}+g_0x_{12}$
<i>clk</i>	<i>(next clk) R6</i> <i>=R5+R1+R2</i>				<i>(next clk) R7</i> <i>=R6+R3</i>	
1	0				0	
2	$g_1x_1+g_2x_2$				g_3x_1	
3	$g_1x_1+g_0x_2+g_1x_3+g_2x_4$				$g_1x_1+g_2x_2+g_3x_3$	
4	$g_3x_1+g_2x_2+g_1x_3+g_0x_4+g_1x_5+g_2x_6$				$g_1x_1+g_0x_2+g_1x_3+g_2x_4+g_3x_5$	
5	$g_3x_3+g_2x_4+g_1x_5+g_0x_6+g_1x_7+g_2x_8$				$g_3x_1+g_2x_2+g_1x_3+g_0x_4+g_1x_5+g_2x_6+g_3x_7$	
6	$g_3x_5+g_2x_6+g_1x_7+g_0x_8+g_1x_9+g_2x_{10}$				$g_3x_3+g_2x_4+g_1x_5+g_0x_6+g_1x_7+g_2x_8+g_3x_9$	
7	$g_3x_7+g_2x_8+g_1x_9+g_0x_{10}+g_1x_{11}+g_2x_{12}$				$g_3x_5+g_2x_6+g_1x_7+g_0x_8+g_1x_9+g_2x_{10}+g_3x_{11}$	
8	$g_3x_9+g_2x_{10}+g_1x_{11}+g_0x_{12}+g_1x_{13}+g_2x_{14}$				$g_3x_7+g_2x_8+g_1x_9+g_0x_{10}+g_1x_{11}+g_2x_{12}+g_3x_{13}$	

3.3 Generalization for Even- and Odd-tap Filters

The proposed architecture uses different resources for even and odd number of filter taps. The odd number-tap filter can reduce some multipliers of

coefficients, whereas the even number-tap filter can reduce some of the adders. For example, the 1-D DWT 4-tap filter is designed with 4 multipliers, 3 adders and 6 registers, as shown in Fig. 3.2. The detailed comparison results in general taps are listed in Table 3.2. The even number-tap filter is designed in the same way as 4-tap DWT, such as [16].

Table 3.2 The used resources for even and odd number taps filter.

Tap	Coefficients	Multiples	Adders	Registers
Even number	2	2	2	4
	4	4	3	6
	6	6	5	9
	8	8	7	12
	10	10	9	15
Odd number	3	2	2	4
	5	3	4	6
	7	4	6	8
	9	5	8	10
	13	7	12	14

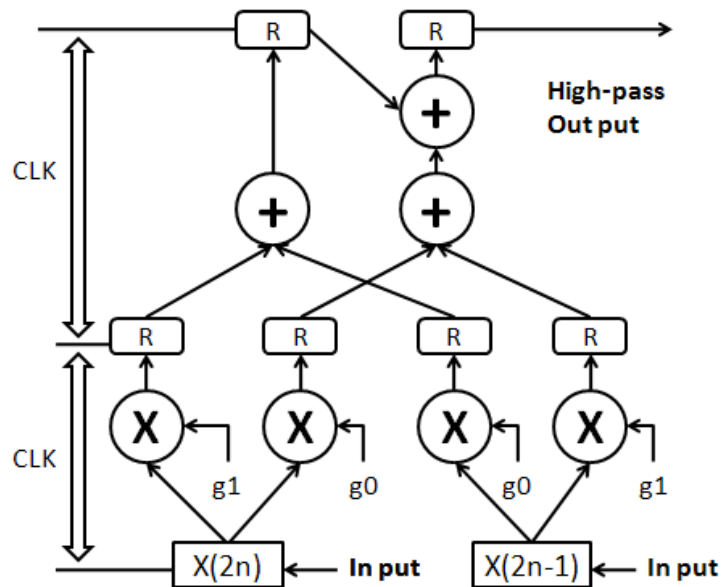


Fig. 3.2 The proposed 1-D DWT with 4-tap (even number) filter.

3.4 High-Speed Performance with RAG

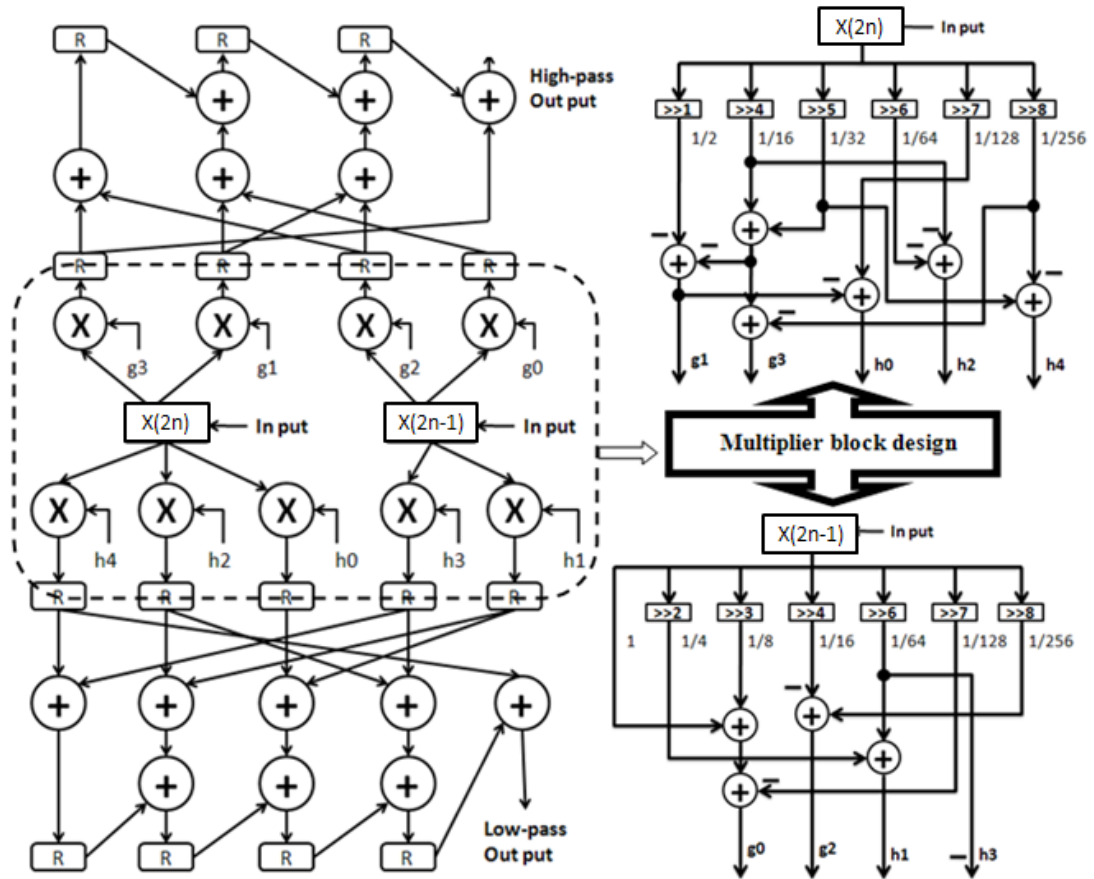


Fig. 3.3 Proposed 1-D DWT by 9/7 taps filter and its multiplier block design.

When performing operation scheduling, an important decision is the selection of a clock cycle to schedule the operations into different states. A bad choice of the clock cycle could adversely affect the performance and the cost of the final design. In Fig. 3.1, the proposed structure shows a two-stage pipeline with the pipeline stage delay (the inverse of throughput) constraint of two adders at every clock. In Fig. 1.4, the lifting structure shows one-stage pipeline with the pipe stage delay constraint of one multiplier plus two adders at every clock.

As explained in section 2.5, the proposed architecture can be further optimized by using the reduce adder graph (RAG) algorithm [4] as shown in the right side of Fig. 3.3. The adaptability of RAG algorithm to the proposed architecture is excellent, but it to the lifting scheme is not efficient. Therefore, the proposed

architecture with RAG can use the faster clock than the lifting scheme, the details of the RAG architecture for 1-D DWT 9/7-tap filter are shown in Table 4.1 and 4.2.

3.5 Proposed ZXY 2-D DWT Architecture

Implementation of the ZXY row processor (RP) is as straightforward as Lifting's Fig. 2.19 (a). The detail was detailed in section 2.61. Actually, the ZXY column processor (CP) using the same way as lifting's, except the size of buffer they used. For ZXY 7-tap filter architecture shown in Fig. 3.4, buffer space for only three rows is required, so the 9/7-tap filter, buffer space for only seven there rows is required as lifting scheme.

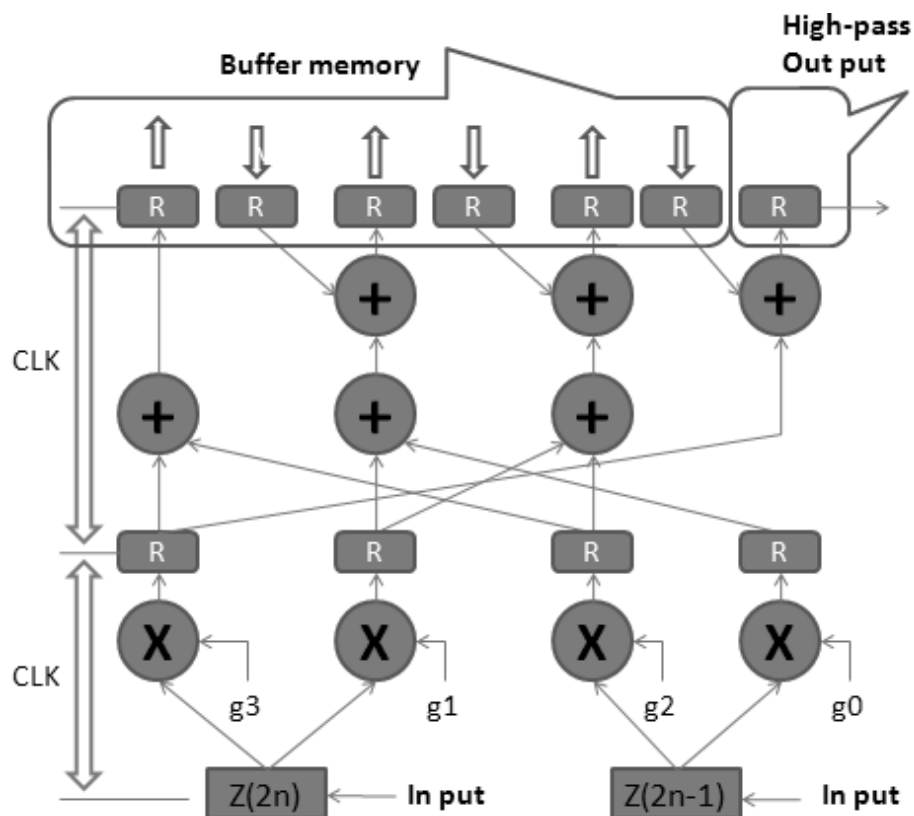


Fig. 3.4 ZXY parallel architecture for column processing (7-tap).

Chapter 4. Performance Analysis and Comparison

In order to show better comparison, we selected 9/7-tap and 5/3 -tap DWT filters for the comparison objects. Fig. 3.9 shows the complete proposed architecture for the 9/7-tap filter with high-pass and low-pass filters. The experimental results on two examples demonstrate the proposed architecture achieved more efficient than the Daubechies and the lifting scheme architectures as shown in Table 4.1.

Our efficient pipeline architecture for 1-D DWT is as simple as the Daubechies architecture, and has many benefits as below. Firstly, ZXY architecture has the same separability as the Daubechies architecture, which means the high-pass FIR filter and the low-pass FIR filter of them can perform independently, but lifting scheme can not. Secondly, ZXY architecture is pipeline design, so that the hardware complexity was reduced with little resources overhead. At last, ZXY is the better applicability. There are two options for clock cycle selection. The long clock cycle (the delay of one multiplier plus two adders) is for register reduction, and the short clock cycle (the maximum delay of one multiplier or two adders) for high-speed performance. Moreover, the proposed architecture can be designed as any kind of taps filter as shown in Table 4.1.

Table 4.1 Performance Comparison for 1-D DWT 9/7 and 5/3-tap filters.

Architecture		M	A	R	clk cycle	Latency	Shift	
Daubechies	5/3	8	6	5	4A+1M	4clk	--	
	9/7	16	14	9	8A+1M	4clk	--	
Lifting	5/3	4	4	8	2A+1M	5clk	--	
	9/7	6	8	14	2A+1M	7clk	--	
ZXY	HS	5/3	5	6	5	2A+1M	3clk	--
	AR	5/3	5	6	10	2A	4clk	--
	HS	9/7	9	14	9	2A+1M	3clk	--
	AR	9/7	9	14	18	2A	4clk	--
(RAG)Lifting	9/7	0	22	14	2A+1M	7clk	16	
(RAG)ZXY	9/7	0	24	18	2A	4clk	13	

HS: High Speed type; AR: Area Reduction type

L: Latency; R: Register; S: Shift

A: One Adder's Delay Time or Adders

M: One Multiplier's Delay Time or Adders

Latency shown in this table is for low-pass output. Usually, high-pass output comes early than low-pass one or two clock cycle.

Fig. 4.1, 4.2 and 4.3 are Daubechies, Lifting and ZXY 9/7-taps filter simulation results, respectively. They are implemented in 11-bit fixed-point input and output, by Xilinx Spartan3E XC3S1200, FG320, Speed: -4, and software: Xilinx ISE 11.1. They used the same the discrete input signals, and got the same results, except deviation. They are using 11 bit signed fixed point for computation.

In Fig. 4.1, Daubechies filter is improved by two stage pipelining, its minimum clock cycle is dominated by one multiplier latency (18.776ns), because eight-adder latency is shorter than one-multiplier latency. However, we set the clock to be 20ns for easy to show in Fig. 4.1. Because one register has been used after each multiplier to modify Daubechies architecture for higher-speed performance, which is shown in Fig. 2.1

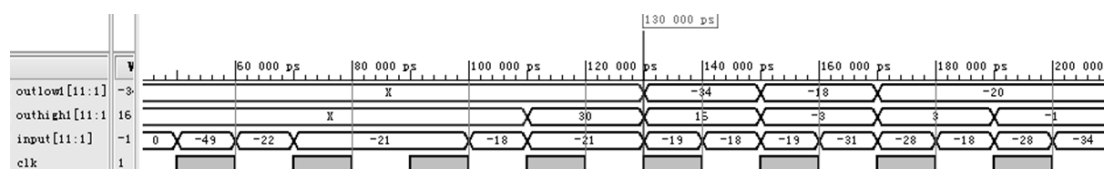


Fig. 4.1 Simulation for Daubechies DWT by 9/7-tap filter.

In Fig. 4.2, Lifting filter's minimum clock cycle depends on one multiplier and two-adder latency (23.805ns). So we set clock cycle to 24ns for this circuit. The architecture in Fig. 2.2(b) directly connects the multiplier which is the one before high-pass output, so high-pass output latency is shorter than low-pass two clock cycle. The first high-pass output is number 26, and the first low-pass output is number -33.

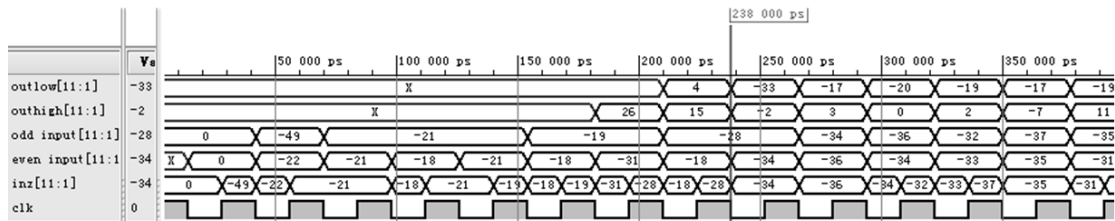


Fig. 4.2 Simulation for Lifting DWT by 9/7-tap filter.

In Fig. 4.3, the proposed filter's minimum clock cycle depends on only one multiplier latency (10.233ns) when it used RAG for multiplier block design. We set the clock cycle is 12ns in this case, whose circuit structure is shown in Fig. 3.9.

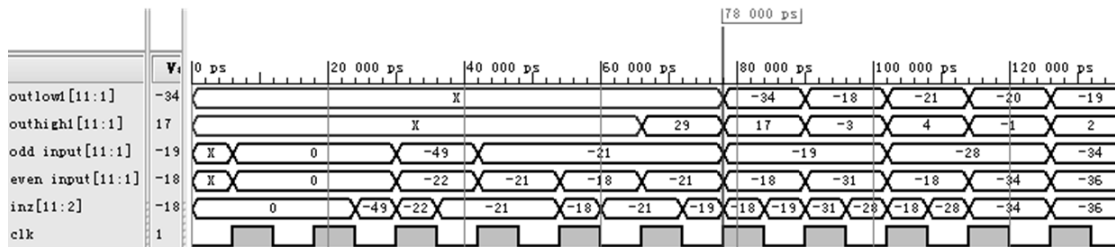


Fig. 4.3 Simulation for ZXY DWT by 9/7-tap filter.

Table 4.2 Performance Comparison for DWT 9/7-tap filter by Xilinx ISE 11.

Circuit	Slices	SFFs	LUTs	Bit	MCC	LHP	LLP	Figure
Daubechies	502	274	905	11	18.776ns	4 clk	3 clk	Fig. 10
Lifting	323	165	533	11	23.805ns	7 clk	5 clk	Fig. 11
	490	240	835	16	--			--
ZXY	344	216	649	11	18.776ns	4 clk	3 clk	--
	632	321	1190	16	--			--
ZXY (RAG)	152	204	269	11	10.233ns	4 clk	3 clk	Fig. 12
	307	321	583	16	--			--

MCC: minimum clock cycle;

FFs: Slices Flip Flops;

LHP/LLP: latency for High-pass/Low-pass;

Chapter 5. Conclusions

An efficient pipeline architecture 1-D DWT based on the Daubechies architecture has been proposed. The proposed DWT is composed of two independent FIR filters: a high-pass FIR filter and a low-pass FIR filter. Each FIR filter is a two-stage pipeline whose fastest clock cycle is only either two adders delay or one multiplier delay. The proposed architecture can achieve the higher speed than lifting because the clock cycle is one multiplier delay or two adders delay, this is in contrast to one multiplier delay plus two adders delay found in the lifting architecture. The proposed architecture still be larger area for the implementation using fixed-point adder and multiplier, but is almost the same area for the implementation using the reduce adder graph (RAG) algorithm. The proposed architecture can be further generalized to implement as any even- and odd- taps of filters.

Our efficient pipeline architecture for DWT has many benefits as follows:

- It is a pipeline architecture that works efficiently without complicated control.
- It can be adapted to any-tap of DWT filters.
- It is a kind of transposed FIR structure. Multipliers have the same input that is good for adapting RAG algorithm for size reduction.
- It is a two-input structure which is the same type as Lifting structure, but shorten latency and shorten clock cycle with little resource overhead.
- It is available to be regarded as two independent FIR filters: a high-pass filter FIR filter and a low-pass filter.

References

- [1] David S. Taubman and Michael W. Marcellin, JPEG2000: Image Compression Fundamentals, Standards and Practices, Kluwer Academic Publishers, Norwell, Massachusetts, 2002..
- [2] S. V. Silva and S.Bampi, "Area and Throughput Trade-Offs in the Design of Pipelined Discrete Wavelet Transform Architectures," IEEE Proceedings of the Design, Automation and Test in Europe Conference and Exhibition. pp.1530-1591, 2005
- [3] Ingrid Daubechies and Wim Sweldens, "Factoring Wavelet Transforms into Lifting Steps", J. Fourier Anal. Appl., vol.4, no.3, pp.247-269, 1998.
- [4] A. Dempster, M. Macleod: "Comments on "Minimum Number of Adders for Implementing a Multiplier and Its Application to the Design of Multiplierless Digital Filters"," IEEE Transactions on Circuits and Systems II 45, pp.242-243, 1998.
- [5] R. M. Rao and A. S. Bopardikar, Wavelet Transforms: Introduction to Theory and Application. Addison-Wesley, MA, 1998.
- [6] M. Anotonini, M.Barlaud, P. Mathieu, and I. Daubechies, "Image Coding Using Wavelet Transform," IEEE Trans. on Image Processing, Vol. 1, No. 2, pp. 205-220, April 1992.
- [7] David Salomon, Data Compression: the Complete Reference, Springer-Verlag, New York, 2nd edition, 2000.
- [8] ITU-T Recommendation T.800, "JPEG 2000 IMAGE CODING SYSTEM - Coding of Still Pictures", V1.0, 16 March 2000.
- [9] A.S. Motra, P.K. Bora, and I. Chakrabarti, "An Efficient Hardware Implementation of DWT and IDWT", Conference on Convergent Technologies for Asia-Pacific Region (TENCON 2003), vol.3, pp.15-17 October 2003.
- [10]S. Masud, and J.V. McCanny, "Reusable Silicon IP Cores for Discrete Wavelet Transform Applications", IEEE Transactions on Circuits and Systems-I: Regular Papers, vol. 51, no. 6, June 2004.

- [11]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 Transactions on Circuits and Systems for video technology, vol. 13, no.9, September 2003.
- [12]Tinku Acharya and Ping-Sing Tsai, "JPEG2000 Standard for Image Compression." John Wiley & Sons, Inc, New Jersey, 2005
- [13]W. Sweldens, "The Lifting Scheme: A Custom-Design Construction of Biorthogonal Wavelets," Applied and Computational Harmonic Analysis, Vol. 3, NO. 15, pp. 186-200, 1996.
- [14]Maurizio Martina, Guido Masera, "Low-Complexity, Efficient 9/7 Wavelet Filters VLSI Implementation," IEEE transactions on circuits and systems. Vol.53, no.11, 2006.
- [15]Youn-Hong Kim, Chung-Hwa Kim, and Kang-Hyeon Rhee, "FPGA Implementation of Sub-band Image Encoder Using Discrete Wavelet" IEEE TENCON, pp. 1335-1338, 1999.
- [16]Tze Yun Sung, "Design of Multiplierless and High-Performance DWT and IDWT Architectures Using 4-taps Daubechies Filters", Special Issue on Applied Mathematics and Microelectronics Engineering. vol.6, no.3, pp.69-79, 2008.
- [17]T. ACHARYA, C. CHAKRABARTI, "A Survey on Lifting-based Discrete Wavelet Transform Architectures," Journal of VLSI Signal Processing. pp. 321-339, 2006.
- [18]C. J. Lian, K. F. Chen, H. H. Chen, and L. G. Chen, "Lifting Based Discrete Wavelet Transform Architecture for JPEG2000," in IEEE International Symposium on Circuits and Systems, Sydney, Australia, pp. 445-448, 2001.
- [19]J.M. Jou, Y.H. Shiau, and C.C. Liu, "Efficient VLSI Architectures for the Biorthogonal Wavelet Transform by Filter Bank and Lifting Scheme," in IEEE International Symposium on Circuits and Systems, Sydney, Australia, pp. 529-533. 2001.
- [20]W.H. Chang, Y.S. Lee, W.S. Peng, and C.Y. Lee, "A Line-Based, Memory Efficient and Programmable Architecture for 2D DWT Using Lifting Scheme," in IEEE International Symposium on Circuits and Systems, Sydney, Australia, pp. 330-333, 2001.

- [21] C.T. Huang, P.C. Tseng, and L.G. Chen, "Flipping Structure: An Efficient VLSI Architecture for Lifting-Based DiscreteWavelet Transform," in IEEE Transactions on Signal Processing, pp. 1080–1089, 2004.
- [22] K. Andra, C. Chakrabarti, and T. Acharya, "A VLSI Architecture for Lifting-Based Forward and InverseWavelet Transform," IEEE Trans. of Signal Processing, vol. 50, no. 4, pp. 966–977, 2002.
- [23] M. Vishwanath, "The Recursive Pyramid Algorithm for the DiscreteWavelet Transform," in IEEE Transactions on Signal Processing, vol. 42, pp. 673–676, 1994.
- [24] H. Liao, M.K. Mandal, and B.F. Cockburn, "Novel Architectures for Lifting-Based Discrete Wavelet Transform," in Electronics Letters, vol. 38, no. 18, pp. 1010–1012, 2002.
- [25] H. Liao, M.K. Mandal, and B.F. Cockburn, "Efficient Architectures for 1-D and 2-D Lifting-BasedWavelet Transform," IEEE Transactions on Signal Processing, vol. 52, no. 5, pp. 1315–1326, 2004.
- [26] P.-Y. Chen, "VLSI Implementation for One-DimensionalMultilevel Lifting-BasedWavelet Transform," IEEE Transactions on Computers, vol. 53, no. 4, 2004.
- [27] Uwe Meyer-Baese, "Digital Signal Processing with Field Programmable Gate Arrays," Third Edition, Springer-Verlag Berlin Heidelberg, pp. 165-192, 2007.
- [28] S. Barua, J. E. Carletta, K. A. Kotteri, A. E. Bell, "An Efficient Architecture for Lifting-based Two-Dimensional Discrete Wavelet Transforms," GLSVLSI'04, Boston, Massachusetts, USA, April, 2004.

VITAE

Name Mr. Zhang Xiaoyin

Student ID 5010120148

Educational Attainment

Degree	Name of Institution	Year of Graduation
Bachelor of Engineering	JiangXi University of Science and Technology	2006

List of Publication and Proceedings

- [1] X. Y. Zhang, N. Jindapetch, "An Efficient VLSI Architecture for Discrete Wavelet Transform based on the Daubechies Architecture," *Filters and Data Conversion Circuits of the ECTI International Conference (ECTI-CON 2010)*, pp. 258, Chiang Mai, Thailand, May 2010.
- [2] X. Y. Zhang, N. Jindapetch, "An Efficient VLSI Architecture for the Rader Algorithm based DFT Prime-Factor," *Filters and Data Conversion Circuits of the ECTI International Conference (ECTI-CON 2010)*, pp. 257, Chiang Mai, Thailand, May 2010.